МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СЕВЕРО-КАВКАЗСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ

Институт цифровых технологий
Кафедра «Цифровая инженерия и сетевые технологии»



С. Х. Биджиева

УНИФИЦИРОВАННЫЙ ЯЗЫК МОДЕЛИРОВАНИЯ UML

УЧЕБНОЕ ПОСОБИЕ
для обучающихся 3 курса по направлению подготовки
09.03.04 Программная инженерия

Черкесск, 2026

УДК 004:519.7/8
ББК 32.973:22.18
Б 59

Рассмотрено на заседании кафедры «Цифровая инженерия и сетевые технологии»
Протокол № 1 от «06» 09. 2025 г.
Рекомендовано к изданию редакционно-издательским советом СКГА.
Протокол № 28 от «28» 09. 2025 г.

Рецензенты:
Кумратова А. М. – доктор экономических наук**,** доцент, заведующий кафедрой «Информационные системы и технологии» Института Цифровых Технологий Северо-Кавказской Государственной Академии;
Лайпанов М. З. – кандидат физико-математических наук, доцент, заведующий кафедрой «Физика» Карачаево-Черкесского Государственного Университета им. У. Д. Алиева.

Б59 **Биджиева, С. Х.** Унифицированный язык моделирования UML: учебное пособие для обучающихся 3 курса по направлению подготовки 09.03.04 Программная инженерия / С. Х Биджиева. – Черкесск: БИЦ СКГА, 2026.– 108 с.

Учебное пособие предназначено для обучающихся по направлению подготовки 09.03.04 «Программная инженерия» для аудиторных и самостоятельных занятий по предмету «Унифицированный язык моделирования UML». В пособии описываются основные элементы нотации диаграмм UML, на конкретных примерах рассматривается процесс визуального моделирования программных средств с применением CASE – технологии StarUML.

УДК 004:519.7/8
ББК 32.973:22.18

© Биджиева С. Х., 2026
ФГБОУ ВО СКГА, 2026

ОГЛАВЛЕНИЕ

Введение 1
Глава I. Объектно-ориентированный анализ и проектирование 5
§1. Методология объектно-ориентированного программирования 5
§2. Методология объектно-ориентированного анализа и проектирования 7
§3. Основные этапы развития языка UML 8
§4. Общая характеристика моделей объектно-ориентированного анализа и проектирования 10
§5. Программные инструменты визуального моделирования 12
Раздел 2. UML - нотация и метамодель 16
§1. Нотация языка UML 16
§2. Диаграмма вариантов использования 22
§2.1 Диаграмма вариантов использования: понятие, задачи и требования 22
§2.2 Отношения на диаграмме вариантов использования 23
§2.3 Дополнительные обозначения языка UML для бизнес-моделирования 25
§2.4 Рекомендации по разработке диаграмм вариантов использования 26
§2.5 Создание нового проекта в StarUML 27
§3. Диаграммы взаимодействия (Interaction diagrams) 34
§3.1 Диаграмма последовательности (Sequence-диаграммы) 34
§ 3.1 Диаграмма кооперации 45
§ 3.2 Создание диаграммы кооперации 46
§4. 1 Диаграммы состояний 51
§4.2 Создание диаграммы состояний 55
§5.1 Диаграмма активности (Activity diagram) 60
§5.2 Построение диаграммы активности для варианта использования «Выполнить поставку Заказа 62
§ 6.1 Создание диаграмм классов 65
§ 6.2 Пакеты в языке UML 69
§ 6.4 Уточнение методов и свойств классов 72
§ 6.5 Типы связей между классами UML 77
§ 7. Диаграмма компонентов и особенности ее построения 84
§ 8. Диаграмма развертывания, особенности ее построения 90
§ 8.1 Диаграмма развертывания 90
§ 8.2 Рекомендации по построению диаграммы развертывания 95
§ 9. Рекомендации по графическому изображению диаграмм языка UML 96
Глоссарий 101
Список использованных источников 98

ВВЕДЕНИЕ

В настоящее время, без преувеличения, компьютерные и информационные технологии можно назвать наиболее динамично развивающейся областью современных знаний, объединяющей новейшие достижения науки и техники. Появление новых моделей процессоров и компонентов, операционных систем и версий программного обеспечения происходит на фоне постоянно возрастающей сложности не только отдельных физических и программных компонентов, но и лежащих в их основе концепций. Развитие и совершенствование программных продуктов означает, что, несмотря на их постоянное улучшение и модификацию, необходимо сохранять единый стиль в различных версиях программ.

Сложность создания современных приложений на начальных этапах проекта обычно оценивается значительно ниже фактических затрат на рабочую силу, что приводит к незапланированным расходам и задержкам в окончательном завершении программы. По мере развития разработки приложения функциональные требования клиента меняются, что еще больше задерживает завершение работы программистов. Увеличение количества программ требует привлечения внештатных программистов, что, в свою очередь, требует дополнительных ресурсов для координации их работы. В разработке и внедрении современных программных средств задействовано большое количество специалистов различной квалификации, и для них унифицированная архитектура и функциональность представляют значительные трудности.

Таким образом, все эти особенности требуют моделирования структуры и функционирования программных систем до написания соответствующего кода. Создание предварительной модели программной системы является необходимым условием для успешного завершения проекта.

Модель – это абстрактное представление физической системы на определенном языке с помощью специальных графических средств, рассматриваемое с определенной точки зрения.

На основе общих принципов системного анализа одна и та же физическая система может быть представлена несколькими моделями. Назначение конкретной модели системы определяется характером решаемой задачи. Главное требование к модели программной системы заключается в том, что она должна быть понятна клиенту и всем членам проектной группы, включая бизнес-аналитиков и программистов. Такая нотация была разработана благодаря усилиям группы экспертов ведущих производителей программного и аппаратного обеспечения, что привело 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.

Глава I. Объектно-ориентированный анализ и проектирование
§ 1. Методология объектно-ориентированного программирования

Объектно-ориентированная методология программирования стала альтернативой процедурной или алгоритмической организации структуры кода программы. Во второй половине 80-х годов прошлого столетиям традиционные методы процедурного программирования не справлялись с возрастающей сложностью разрабатываемых программ, а также с повышением их надежности, в связи с чем возникла острая необходимость в новой методологии программирования, которая решила бы все эти проблемы. Такой методологией стало объектно-ориентированное программирование (ООП).

Объектно-ориентированное программирование (ООП) – это набор принципов, технологий и инструментов для создания программных систем, основанных на архитектуре взаимодействия объектов.

В объектно-ориентированной методологии, в отличие от процедурно-ориентированной декомпозиции, в качестве структурных единиц программы рассматриваются не процедуры и функции, а классы и объекты с соответствующими свойствами и методами. Основные принципы ООП - это абстракция, наследование, инкапсуляция и полиморфизм.

Абстракция (abstraction) – характеристика сущности, которая отличает ее от других сущностей. Абстракция определяет границу представления соответствующего элемента модели и применяется для определения фундаментальных понятий ООП, таких как класс и объект.

Класс представляет собой абстракцию совокупности реальных объектов, которые имеют общий набор свойств и обладают одинаковым поведением. Объект в контексте ООП рассматривается как экземпляр соответствующего класса.

Объекты, которые не имеют идентичных свойств или не обладают одинаковым поведением, по определению, не могут быть отнесены к одному классу.

Классы могут быть организованы в иерархическую структуру. Иерархия понятий строится следующим образом. Наиболее общее понятие или категория имеет наибольший объем и, следовательно, наименьшее содержание. Это высший уровень абстракции для данной иерархии. Затем это универсальное понятие уточняется или конкретизируется, то есть его объем уменьшается, а содержание увеличивается. В схеме иерархии появляется менее общее понятие, которое будет располагаться на более низком уровне, чем исходное. Конкретизация понятий может продолжаться до достижения понятия самого низкого уровня; дальнейшая конкретизация в этом контексте либо невозможна, либо не имеет смысла.

В методологии объектно-ориентированного программирования принцип наследования рассматривается как возможность создавать иерархии классов: классы потомки могут наследовать данные и методы родительских

классов. Исходя из этого некоторые классы выступают наиболее абстрактными и общими по отношению к другим классам.

В качестве примера можно привести класс Животные, содержащий методы и средства для всех животных. Класс Кошка наследует эти методы и средства, также имеет свои собственные уникальные методы и средства.

Следующий принцип ООП – инкапсуляция. Инкапсуляция характеризует сокрытие отдельных деталей внутреннего устройства классов от внешних по отношению к нему объектов или пользователей.

При взаимодействии с объектом класса клиенту не требуется знать, как реализован тот или иной элемент класса. Конкретное применение свойств и методов, присущих классу и определяющих его поведение, является задачей самого класса. Более того, отдельные свойства и методы класса могут быть невидимы за его пределами, что связано с основной идеей применения различных категорий видимости к элементам класса.

Инкапсуляция возникла как результат деления модулей в некоторых языках программирования на две части: интерфейс и реализацию. При этом интерфейсная часть (фронтенд) модуля описывает все объявления функций и процедур, а также, возможно, типы данных, существующие вне модуля. Эти процедуры и функции являются методами обслуживания внешних клиентов. Другая часть модуля, называемая реализацией или бэкендом, содержит программный код, определяющий конкретные способы выполнения процедур и функций, объявленных в интерфейсе.

В качестве примера реализации принципа инкапсуляции можно привести работу банкомата: личные деньги (баланс) – это приватное поле, интерфейс банкомата – публичные методы.

Полиморфизм как один из принципов методологии объектно-ориентированного программирования предполагает, что, действия, которые выполняются одними и теми же методами, зависят от того к какому классу относится тот или иной метод и, как следствие, могут различаться.

В качестве примера рассмотрим работу мессенджера, действие «Отправить сообщение». Можно отправить разные типы сообщений:

– текст – проверяется длина, кодируется в UTF-8, добавляется в очередь отправки;

– фото – сжимается, конвертируется в JPEG, загружается на сервер, генерируется превью;

– видео – проверяется размер, конвертируется в MP4, создается постер;

– локация – преобразуются координаты, формируется ссылка на карты;

– стикер – берется ID стикера, проверяется доступность в пакете.

Пользователь просто нажимает «Отправить», а система для каждого типа контента выполняет разные операции.

Важнейшим моментом в развитии объектно-ориентированной методологии стало понимание того, что процесс написания программного

кода можно отделить от процесса проектирования структуры программы. Прежде чем начинать программировать классы, их свойства и методы должны быть определены сами классы. Кроме того, необходимо ответить на следующие вопросы: сколько и какие классы необходимо определить для решения задачи, какие свойства и методы необходимы для придания классам необходимого поведения, а также установить связи между классами. Этот набор задач не связан с написанием кода, а с общим анализом требований к будущей программе, а также с анализом конкретной предметной области, для которой разрабатывается программа. Все эти обстоятельства привели к появлению специальной методологии, называемой объектно-ориентированной методологией анализа и проектирования (ООАП).

§ 2. Методология объектно-ориентированного анализа и проектирования

Необходимость анализа предметной области перед написанием программы возникла при разработке крупномасштабных проектов. Процесс создания базы данных существенно отличается от написания программного кода для решения вычислительной задачи. Поэтому при проектировании базы данных необходимо предварительно разработать концептуальную модель, отражающую общие взаимосвязи предметной области и конкретные детали организации соответствующей информации**.**

Предметная область как часть реального мира, имеющая существенное значение к процессу функционирования программы, включает в себя только те объекты и взаимосвязи между ними, которые необходимы для описания требований и условий решения определенных задач ().

Определение исходных или основных компонентов предметной области, необходимых для решения конкретной задачи, обычно представляет собой незначительную проблему. Сложность этой проблемы заключается в неформальном характере процедур или правил, которые могут быть применены для этой цели. Более того, эта работа должна выполняться совместно с экспертами или специалистами, хорошо разбирающимися в данной предметной области. Например, при разработке базы данных для использования в крупной строительной компании, в проектировании концептуальной модели базы данных должны принимать участие сотрудники организации, которые хорошо знают специфику и нюансы этой предметной области. И как решение данной проблемы появилась новая методология объектно-ориентированного анализа и проектирования.

Объектно-ориентированный анализ и проектирование (ООАП) – это технология разработки программных систем, основанная на объектно-ориентированной методологии представления предметной области в виде объектов как экземпляров соответствующих классов.

Методология ООАП тесно связана с концепцией автоматизированной разработки программного обеспечения (Computer Aided Software Engineering,

CASE) и их связь исторически была очень важной для эволюции разработки программного обеспечения.

ООАП как методология и набор концепций отвечает на вопрос «Что делать?» (как мыслить, как декомпозировать систему диаграмм, какие модели создавать). Case – это инструментарий и технология, которая поддерживает процесс разработки отвечает на вопрос «С помощью чего делать?» (какие инструменты использовать для создания диаграмм, генерации кода, управления требованиями).

Case – инструменты стали естественной средой для применения и визуализации и методологии ООАП: диаграммы Unified Modeling Language (UML) стали стандартным языком для ООАП, Case-технологии предоставляли редакторы для создания этих диаграмм, давали возможность автоматизации рутинных операций (автоматически генерировать каркас кода на целевом языке программирования). Современные Case-инструменты могут анализировать существующий исходный код и автоматически строит по нему диаграммы, что позволяет поддерживать модель в актуальном состоянии при изменении в коде и наоборот, инструменты могут проверять созданные ОО – модели на непротиворечивость, полноту, соответствие правилам UML и методологии.

Надо отметить, что настоящее время термин «Case» используется реже. Сейчас больше говорят об «инструментах моделирования» (Modeling Tools), «средах MDA (Model-Driven Architecture)» или просто о возможностях «визуального моделирования». Но от этого суть использования этих средств – автоматизация работы с моделями – не изменилась. Можно сказать, ООАП предоставили мощную концептуальную основу для моделирования сложных систем, Case- технологии дали практический инструментарий для реализации этого подхода, повысив производительность, стандартизацию и качество процесс разработки.

§ 3. Основные этапы развития языка UML

История развития языка UML началась в 90-ые годы прошлого столетия. До появления UML существовало множество конкурирующих методологии и нотаций для объектно-ориентированного анализа и проектирования, среди которых можно выделить:

– метод Boocha (Grady Booch): основной акцент в этой методологии делался на проектирование и архитектуру;

– object modeling technique (OMT) Джеймса Румбаха сильной стороной, которой являлись – анализ и моделирование данных**;**

– object–oriented software engineering (OOSE) Айвара Джекобсона, который ввел ключевые концепции прецендентов (use cases) и вариантов использования.

Эти методы были достаточно популярны, но разработчикам приходилось выбирать между ними или пытаться совмещать, что вызывало

путаницу, и в итоге привело к осознанию необходимости унификации этих методов. Совместная работа, которая началась осенью 1994 года, была направлена на изучение и объединение всех преимуществ этих объектно-ориентированных методов с целью усиления их преимуществ. Одновременно Г. Буч и Й. Румбах сосредоточили свои усилия на полной интеграции результатов своей работы. Проект под названием Unified Method версии 0.8 был разработан и опубликован осенью 1995 года. Осенью того же года к ним присоединился главный технолог компании Objectory AB А. Джекобсон, чтобы объединить свой метод OOST с методами Гради Буча и Джеймса Румбаха. Трио, известное как «Три Амиго» (Tht Three Amigos), становится движущей силой создания унифицированного языка.

В 1996 году они представили первую публичную версию – UML 0.9, которая быстро набрала популярность в сообществе. К процессу подключились другие компании (IBM, Oracle, Microsoft и др. Компания Rational Software, совместно с рядом организаций, готовых выделить ресурсы на разработку строгого определения версии языка UML 1.0, сформировала консорциум партнеров UML, в который первоначально вошли такие компании, как Digital Equipment Corp., HP, i-Logix, Intellicorp, IBM, Icon Computing, MCI Systemhouse, Microsoft, Oracle Rational Software, Ti и Unisys. Эти компании поддержали дальнейшую работу над более детальным определением языка.

UML 1.0 и UML 1.1 были представлены сообществу в 1997 году.

В этот период консорциум OMG (Object Management Group), который был создан еще в 1989 году с целью разработки предложений по стандартизации объектных и компонентных технологий, принимает UML 1.1. в качестве официального стандарта. UML превращается в международный стандарт объектно-ориентированного моделирования.

В последующие версии (1.2, 1.3, 1.4, 1.5) вносились в основном косметические исправления, уточнялась семантика, добавлялись новые возможности, но основная структура оставалась неизменной. UML 1.x имел недостатки, особенно в области спецификации архитектуры и поведения сложных систем

Значительные изменения были внесены UML 2.x.:

– улучшенная масштабируемость: стало возможным более детальное и точное моделирование больших и сложных систем;

– структурная декомпозиция: применение составных структур для моделирования внутренней структуры компонентов и классов (Composite Structure Diagram);

– улучшенные диаграммы поведения: диаграмма взаимодействия была разделена на несколько типов: последовательности (Sequence), коммуникация (Communication), взаимодействие обзора (Interaction Overview) и временные (Timing);

– возможности диаграмм состояний (State Machine) и диаграмм деятельности (Activity Diagram) значительно расширились, что сделало их

практически мощным инструментом для моделирования бизнес-процессов и потоков работ.

– формальность и точность: метамодель UML (язык, описывающий сам UML) стала более формальной и строгой, что способствовало улучшению однозначности моделей. (между UML 1.x и UML 2.0https://dit.isuct.ru/Publish_RUP/core.base_rup/guidances/supportingmaterials/differences_between_uml_1_x_and_uml_2_0_CA70F2E6.html, Web Page Template: https://setgetweb.com/p/rational/appdev/manuals/intro/im_uml2.htm)

Версии UML 2.x (2.1, 2.2,... 2.5) продолжали развиваться, внося изменения и уточнения. Важным шагом стала версия UML 2.5 (2015), которая упростила спецификацию и сделала изучение языка более удобным за счет объединения документов.

§ 4. Общая характеристика моделей объектно-ориентированного анализа и проектирования

Язык UML – это общий язык визуального моделирования, предназначенный для спецификации, визуализации, проектирования и документирования программных компонентов, бизнес-процессов и других систем.

Как язык для спецификации UML позволяет идентифицировать все важные аналитические, проектные и прикладные решения, принятые в процессе создания и внедрения программного средства.

UML – как язык для визуализации позволяет решить ряд стратегических проблем при разработке программных средств:

– во-первых, проблему чтения текста. В разработке ПС есть некоторые моменты, которые непонятны из текста кода (даже с хорошими комментариями). UML – это графический язык, и поэтому он решает эту проблему.

– во-вторых, проблему потери данных. Если разработчик повредит часть кода и не запишет модель в какой-либо форме, этот код будет потерян навсегда. UML обеспечивает восстановление кода на основе модели.

– в-третьих, проблему интерпретации модели. Команда проекта обычно разрабатывает какой-то внутренний язык (совершенно непонятный извне). UML имеет четко определенную семантику, и поэтому разные разработчики будут интерпретировать модель одинаково.

UML – это язык для проектирования и документирования программных компонентов, бизнес-процессов и других систем. Созданные с её помощью модели могут быть переведены на различные языки программирования. Модель может отображаться на таких языках, как C++, Java, Python и даже в таблицах реляционных баз данных. Возможно прямое и обратное отображение[6,10,12,23]. При создании программного обеспечения создается множество вторичных продуктов, связанных с исполняемым кодом:

системные требования, системная архитектура, проект, исходный код, планы проектирования, тесты, прототипы, версии. UML позволяет фиксировать принятые решения.

Язык UML в первую очередь предназначен для разработки программного обеспечения. Его использование наиболее эффективно в следующих областях:

– корпоративные информационные системы;
– транспорт, включая железнодорожный;
– банковские и финансовые услуги;
– распределенные веб-системы;
– оборонная промышленность, авиация и космонавтика;
– розничная торговля;
– медицинская электроника;
– телекоммуникации;
– наука.

Язык UML – это очень всесторонний и мощный инструмент моделирования, который можно эффективно использовать для построения концептуальных, логических и графических моделей сложных систем для различных целей. Этот язык объединил в себе лучшие качества и опыт методов разработки программного обеспечения, которые успешно применялись в последние годы для моделирования больших и сложных систем.

С точки зрения объектно-ориентированной методологии, достаточно полная модель сложной системы состоит из определенного количества взаимосвязанных представлений, каждое из которых адекватно отражает аспект поведения или структуры системы. В то же время наиболее общие представления сложной системы считаются статическими и динамическими, которые могут быть подразделены на другие, более специфические.

Принцип иерархического построения моделей сложных систем требует рассмотрения процесса создания моделей на разных уровнях абстракции или детализации в рамках устойчивых представлений.

Уровень представления – это способ организации и визуализации модели на одном уровне абстракции, представляющий горизонтальный срез архитектуры модели и наряду с которым существует и вертикальный срез как декомпозиция основных бизнес-процессов.

При этом исходная модель или начальная модель сложной системы имеет наиболее общее описание и относится к концептуальному уровню. Такая модель, называемая концептуальной, строится на начальном этапе проектирования и может не включать многие детали и аспекты моделируемой системы. Последующие модели конкретизируют концептуальную модель, дополняя ее идеями логического и физического уровней.

В целом, процесс объектно-ориентированного анализа и проектирования можно рассматривать как последовательный переход от разработки наиболее общих моделей и представлений концептуального

уровня к более конкретным и детальным описаниям логического и физического уровней. Более того, на каждом этапе ООАП эти модели постоянно дополняются все большим количеством деталей, что позволяет более адекватно отражать различные аспекты специфического функционирования сложной системы[28].

§ 5. Программные инструменты визуального моделирования

В настоящее время существует множество программ для построения UML-диаграмм. Лучшие инструменты для создания UML-диаграмм поддерживают большинство классов UML-диаграмм и совместимы со спецификацией UML 2.0. Выбор инструмента для построения диаграмм зависит от сложности описываемого взаимодействия, стандарта описания, принятого командой, и личных предпочтений. Рассмотрим несколько популярных программ.

Rational Rose – популярная программа создания UML. Это программа включает все необходимые инструменты для точного описания модели, что позволяет получить хороший результат. Этот программный инструмент позволяет значительно ускорить процесс разработки за счет создания UML моделей до начала написания кода программы, что позволяет избежать многих трудностей. В данной программе есть готовые шаблоны, которые помогают в разработке. Частое использование тех или иных шаблонов приводит к их автоматическому отображению в стартовом окне.

В качестве преимущества данной программы можно отметить удобный интерфейс, т.е. работать с программой очень удобно, есть специальные инструменты для работы с XML, также с веб-моделированием.

Edraw UML Diagram – это приложение, которое выбирают многие разработчики и дизайнеры программного обеспечения. Программа полностью совместима с 64- и 32-разрядными платформами Windows от Windows XP до Windows 10.

Во-первых, это программное обеспечение для создания диаграмм предназначено для всех, кому нужна мощная визуализация плана разработки программного обеспечения. Можно в полной мере воспользоваться преимуществами прогрессивных блок-схем и визуальных элементов, проектов планировки помещений или простых электрических схем.

Этот инструмент предоставляет все возможности, которые могут понадобиться большинству пользователей для создания UML-диаграмм. С помощью этого программного обеспечения пользователи могут создавать UML-диаграммы вариантов использования, деятельности, компонентов, последовательностей, пакетов, статических структур, диаграммы состояний и диаграммы взаимодействия. В него входят шаблоны UML, с помощью которых пользователи могут быстро создавать диаграммы.

Пользователи также могут выбирать различные варианты цветовой заливки, линий и теней для вставленных объектов и добавлять изображения в

свои диаграммы. Кроме того, с помощью Edraw пользователи могут создавать модели потоков данных, диаграммы состояний, диаграммы Буча.

Moqups-это онлайн-приложение, с помощью которого пользователи могут создавать диаграммы и вайрфреймы для сложных проектов.

Это программное обеспечение можно использовать для различных целей: для проектирования и визуализации проектов, планирования с помощью диаграмм, создания прототипов и для совместной работы с помощью облачных сервисов. Программа содержит эффективные инструменты для разработки проектов, которые в то же время стимулируют творческий процесс в команде, интуитивно понятную платформу с функцией перетаскивания, которая содержит множество виджетов и смарт-фигур для быстрого добавления.

Таким образом, Moqups предоставляет услуги по созданию качественного планирования проектов с использованием макетов, диаграмм и прототипов любого типа.

Microsoft Visio, входящее в пакет приложений MS Office, является отраслевым стандартом в области создания диаграмм. С его помощью пользователи могут создавать самые разные диаграммы, в том числе классы UML. Данное приложение поддерживает спецификацию UML 2.0 и предоставляет шаблоны для шести типов диаграмм UML. Пользователи Visio Professional могут создавать с помощью этого программного обеспечения диаграммы UML классов, последовательности, баз данных, действий, состояний и вариантов использования. Visio содержит множество вариантов форматирования и оформления. Пользователи могут добавлять в свои диаграммы клипарт, фотографии, выноски, контейнеры, гиперссылки и дополнительные диаграммы. Также данный инструмент позволяет создавать UML-диаграммы на основе импортированного исходного кода и интегрировать в них данные в реальном времени посредством связывания данных.

Существует также онлайн-версия Visio Plan, которая поддерживает типы диаграмм UML 2.5. Вероятно, это не самое экономичное приложение для создания диаграмм UML, но обе версии Visio по-прежнему предлагают обширный набор инструментов для создания диаграмм, с которым могут сравниться лишь немногие другие альтернативы.

Пользователи Visio также могут создавать множество других типов диаграмм, таких как блок-схемы, диаграммы Ганта, сетевые диаграммы, организационные диаграммы и многое другое.

Программное обеспечение UML Designer представляет собой набор графических инструментов для легкого редактирования и визуализации моделей UML. Данное программное обеспечение предлагает удобную среду для графического моделирования с использованием функций Eclipse Modeling. Сервис основан на проекте Sirius, который предлагает типовые структуры для UML-диаграмм.

Кроме того, можно настроить рабочий процесс или диаграммы и объединить UML-модели с предметно-ориентированным моделированием,

чтобы работать с несколькими типами моделирования одновременно. Если быть точнее, то можно одновременно работать над моделями UML и Domain-Specific Language (DSL) и изменять диаграммы в соответствии с конечной концепцией.

В программе есть справочные таблицы, в которых изложены основы работы в программе и на официальном сайте приложения можно найти подробное руководство.

Таким образом, выбрав решение UML Designer, можно настраивать лучшие диаграммы моделирования и создавать рабочие процессы без особого обучения.

Следующий программный инструмент визуального моделирования UMLet – это простой инструмент для разработки, с помощью которого пользователи могут быстро создавать UML-диаграммы. Это программное обеспечение с открытым исходным кодом для всех платформ Windows, которое можно загрузить с веб-сайта UMLet. Надо отметить, что для работы с UMLet требуется Java 1.6, поэтому перед запуском программы необходимо установить Java.

Программное обеспечение поддерживает различные диаграммы UML, такие как диаграммы вариантов использования, последовательности, развёртывания, активности и состояний. Кроме того, в этом приложении есть удобная функция отправки по электронной почте, с помощью которой пользователи могут отправлять диаграммы UML по электронной почте.

ConceptDraw Diagram как программа для создания диаграмм общего назначения, может быть использована для моделирования программного обеспечения. Это одна из лучших альтернатив Microsoft Visio для создания диаграмм. ConceptDraw Diagram совместима с платформами Windows 10, 8 и 7, а также Mac.

Пользователи ConceptDraw Diagram могут создавать практически все типы диаграмм UML. Программа включает в себя широкий набор инструментов для рисования и галереи шаблонов элементов для диаграмм.

Пользователи ConceptDraw 12 могут создавать UML-диаграммы, выбрав один из готовых шаблонов. В программе используется технология «живых» объектов, аналогичная связыванию данных в реальном времени в Visio.

Это приложение также предоставляет широкие возможности для обмена данными: пользователи могут отправлять диаграммы по электронной почте, отображать их в виде презентаций, экспортировать в форматы Adobe Flash или PDF и сохранять в виде HTML для веб-страниц. Кроме того, ConceptDraw полностью совместим с Visio, что является преимуществом для пользователей, которым нужно открывать файлы Visio.

StarUML – программа предлагает множество инструментов для разработки программных моделей. Apple, IBM, Samsung, EA, Amazon и Intel - вот лишь некоторые из известных компаний, которые используют StarUML 3 для разработки программного обеспечения. Приложение совместимо с платформами Windows (от XP до 10), macOS и Linux.

StartUML 3 совместим с диаграммами метамодели UML 2.0. Таким образом, пользователи могут создавать около 11 диаграмм UML, а также диаграммы потоков данных, блок-схемы и диаграммы «сущность-связь» (для проектирования баз данных) и представляет собой интегрированный инструментарий для проектирования архитектуры, анализа, моделирования и разработки программных систем.

Среда моделирования StarUML превосходно настраивается в соответствии с требованиями пользователя и имеет высокую степень расширяемости, особенно в области своих функциональных возможностей.

Благодаря расширениям Java, C++ , Python пользователи StartUML 3 могут генерировать Java, C++ , Python коды на основе своих UML-диаграмм и повторно использовать инженерный код. Программное обеспечение включает в себя несколько удобных функций публикации, которые позволяют пользователям делиться своими моделями с помощью HTML-документов или экспортировать их в PDF для печати. Также, используя эту платформу, вы можете воспользоваться такими расширенными функциями, как поддержка дисплеев Retina, менеджер расширений для сторонних расширений, разработка на основе моделей, быстрое моделирование с помощью сокращённых команд редактирования, тёмная или светлая тема и т. д.

Анализ программных инструментов визуального моделирования показал, что лучшими программами для создания UML-диаграмм общего назначения являются Visio и ConceptDraw. Однако StartUML, Edraw UML Diagram и UMLet 14.3 могут оказаться более выгодными альтернативами для пользователей, которым нужно только создавать UML-диаграммы. В данном пособии мы рассмотрим особенности создания UML-диаграмм в StartUML.

Раздел 2. UML - нотация и метамодель

§ 1. Нотация языка UML

Как графический язык визуального моделирования, UML имеет свою нотацию – принятые обозначения. Нотация обеспечивает семантику языка, способ унификации обозначений визуального моделирования и предоставляет исчерпывающее описание системы, которое относительно легко и свободно воспринимается пользователями.

Моделирование с использованием UML осуществляется путем построения серии пошаговых диаграмм, каждая из которых отражает часть, сторону системы.

UML включает три вида блока, такие как:

– сущности;

– отношения;

– диаграммы.

Основным элементом языка UML выступает сущность.

Все сущности можно разделить на структурные, поведенческие, группирующие, аннотационные.

К структурным сущностям, которые представляют статические части модели относятся: класс (class**),** интерфейс (interface), актѐр (actor), прецедент (use case), компонент (component), узел (node).

Поведенческие сущности включают динамические части UML-модели и представлены такими сущностями как: взаимодействие (interaction), деятельность (activity), автоматы (state machine).

Группирующая сущность представлена пакетами (package) и объединяет семантически связанные элементы в единое целое.

Примечание (note) относится к аннотационной сущности и добавляется к UML-модели для записи специальных уточняющих данных.

Рассмотрим подробно некоторые из этих сущностей**.**

Класс – это описание набора объектов с общими атрибутами, операциями и семантикой (рис.1.1). Графически класс представляется в виде прямоугольника, разделённого на три части, где записываются его имя, атрибуты и операции. Объединение объектов в классы вводит понятие «абстракция» в процесс разработки программного обеспечения.



Рисунок 1.1 – Класс

Цель абстракции – ограничение универсальность, всеобщность понятия (вещи). Абстракция – это анализ объекта с точки зрения его наиболее важных характеристик, которые отличают его от всех других типов объектов и от

других объектов. Абстракция фокусируется на внешних свойствах объекта и позволяет отделить существенные особенности поведения от деталей их реализации.

Надо отметить, что на разных этапах создания ПС разработчики работают с разными классами:

– на этапе анализа ПС классы представляют собой концептуальные абстракции, являющиеся частью предметной области;

– на этапе проектирования – логические абстракции;

– на этапе реализации – это классы языка программирования, выбранные для реализации логических абстракций.

Одна из целей разработки ПС является описание классов/объектов, которые вместе образуют проектируемую систему.

Рассмотрим основные характеристики класса/объекта.

Первая характеристика – уникальное имя, которое позволяет отличать объекты друг от друга.

Вторая характеристика – атрибут как именованное свойство объектов класса.

Каждый атрибут имеет своё значение для каждого объекта.

Среди атрибутов выделяют:

– статические (условное постоянное свойство);

– динамические (переменные условные свойства);

– производные (вычисляемые свойства), значения которых определяются значениями статических и динамических атрибутов объекта.

К третьей характеристике можно отнести состояние объекта как состояние, в котором объект удовлетворяет определённому условию в течение своего существования, выполняет какую-либо деятельность или ожидает определённых событий. Состояние выражается всеми текущими значениями объекта.

Поведение – это характеристика класса/объекта, которая заключается в том, как объект ведет себя и реагирует (то есть, как он изменяет свое состояние). Поведение выражается через состояние объекта и передачу сообщений. Поведение может иметь значение. Понятие сообщения совпадает с понятием операции.

Операция рассматривается как действие, которое объект должен выполнить для реализации своего поведения, или услуга, которую можно запросить у другого объекта. Каждая операция имеет свой собственный объект-получатель, то есть объект, к которому применяется операция. Действие может содержать дополнительные аргументы, которые параметризуют его.

Некоторые операции могут приобретать разные формы в различных классах, такие операции называют полиморфными.

К стереотипным операциям для всех классов относятся следующие операции: − создать и инициировать (create and initialize) объект.

− удалить (delete) объект.

− получить (get…) значение атрибута.

− установить (set…) значение атрибута.

− добавить объект (add…)

– добавить связь с другим объектом.

− исключить объект (remove…)

– исключить связь с другим объектом.

К наиболее распространённым операциям относятся:

− модификатор – операция, которая меняет состояние объекта.

− запрос (селектор) – операция, которая считывает состояние объекта, но не меняет состояние

− итератор – операция, которая позволяет организовать доступ ко всем частям объекта-контейнера для последующего их использования в строго определённой последовательности (например, просмотр очереди).

− преобразование – операция, производящая объект другого типа.

Конкретная реализация операции называется методом.

Следующая сущность, которую мы рассмотрим в рамках данного параграфа актор (рис.1.2). Актор – это любая сущность, взаимодействующая с системой извне.

Акторы не могут контролироваться системой, потому что определяют все то, что находится вне системы.

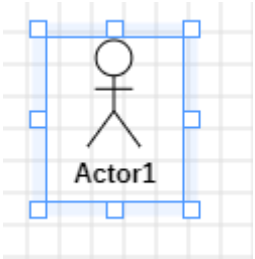


Рисунок 1.2 – Актор

Выделяются четыре основных типа акторов:

1. Физические личности. Например, в банковской системе -это клиенты и обслуживающий персонал.

2. Другая система. Когда другая система выступает как актор, предполагается, что она вообще не изменится. Например, программа электронной почты, форматирующая исходящие сообщения, выступает в роли цензора, удаляя из них неприемлемый контент. Когда появляются данные, требующие обработки, агент ввода начинает асинхронно передавать строки по конвейеру, а агент вывода может аналогичным образом считывать и записывать их в файлы.

3.Время. Время является актором, поскольку оно не контролируется. Если начало любого процесса в системе зависит от него, то время становится актором. Например, в случае неудавшейся транзакции все данные, имевшиеся в системе до ее начала, сохраняются неизменными.

4. Электромеханические устройства. Для дальнейшей обработки система может получать сигналы от этих устройств, а также само может управлять их работой.

Как правило, поскольку актор всегда находится вне системы, его внутренняя структура никак не определяется. Для актора важна только его внешнее представление, поскольку именно как его воспринимает система. Акторы взаимодействуют с системой, отправляя и получая сообщения от сценариев использования. Сообщение – это то, что актор-сервис запрашивает у системы и получает от сервиса. Это взаимодействие может быть выражено через ассоциации между акторами и сценариями использования. Кроме этого, существуют интерфейсы, связанные с акторами, которые определяют, как другие элементы модели взаимодействуют с этими акторами. Более подробно виды ассоциации рассмотрим в следующих параграфах, а сейчас прейдем к рассмотрению следующего блока UML – модели – диаграмм.

В рамках языка UML каждое представление модели сложной системы записывается в виде специальных графических структур, называемых диаграммами.

Диаграмма – это графическое представление набора элементов модели в виде последовательного графа с вершинами и ребрами (дугами), которому присваивается определенная семантика. Каноническая нотация диаграмм является основным инструментом для 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).

Перечень этих диаграмм и их названия являются каноническими в том смысле, что представляют собой неотъемлемую часть графической нотации языка UML. Более того, процесс ООАП неразрывно связан с процессом построения этих диаграмм. При этом совокупность построенных таким образом диаграмм является самодостаточной в том смысле, что в них содержится вся информация, которая необходима для реализации проекта сложной системы.

Эти диаграммы и их имена являются каноническими в том смысле, что они представляют собой неотъемлемую часть графической нотации языка UML. В то же время, процесс объектно-ориентированного анализа и проектирования (ООАП) имеет неразрывную связь с процессом построения этих диаграмм. Комплекс диаграмм, построенных таким образом, является самодостаточным в том смысле, что он содержит всю информацию, необходимую для реализации сложного системного продукта.

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.

В целом интегрированная модель сложной системы в нотации UML может быть представлена в виде совокупности указанных выше диаграмм (рис.1.3)

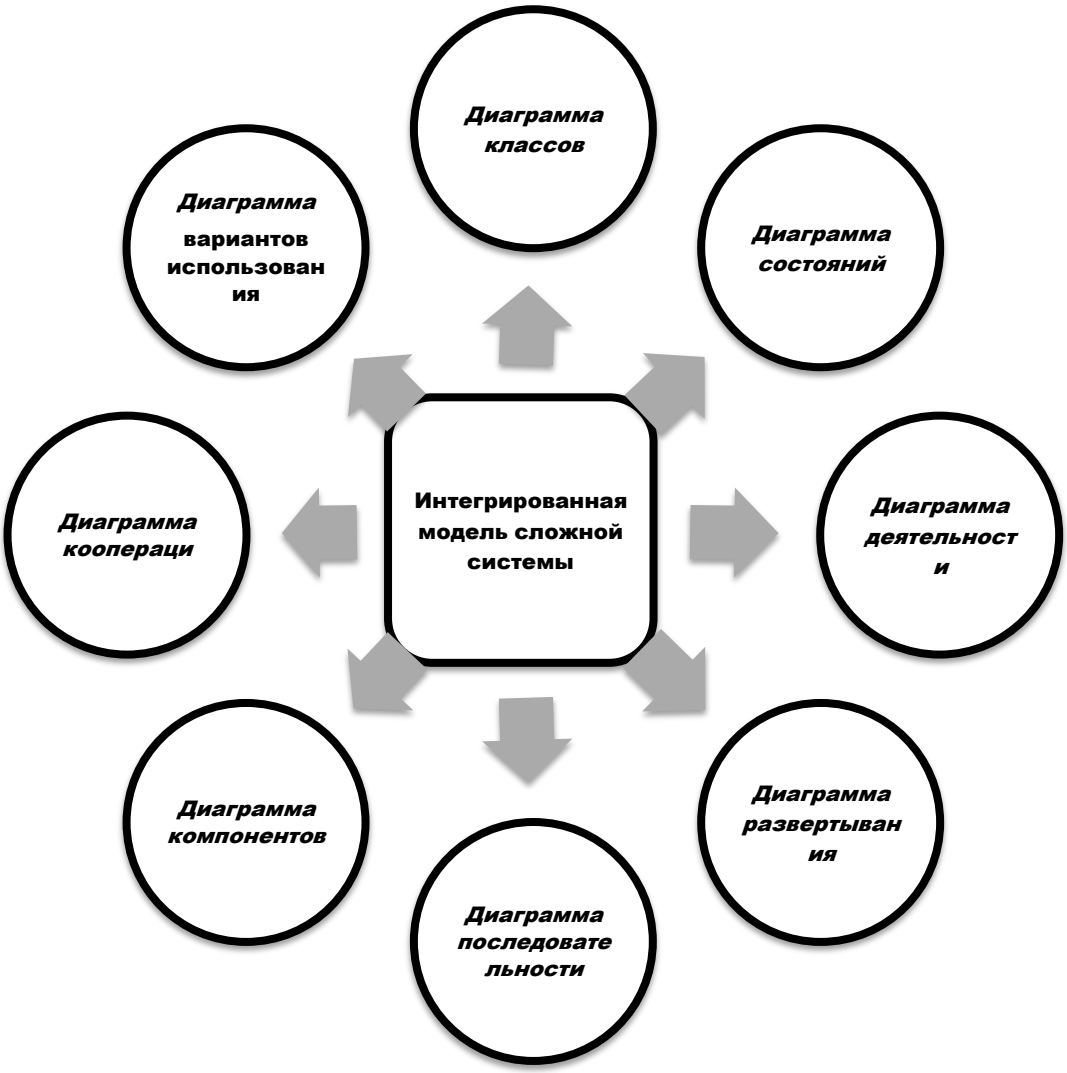


Рисунок 1.3 – Интегрированная модель сложной системы в нотации UML

Помимо графических элементов, определенных для каждой канонической диаграммы, они также могут отображать текстовую информацию, расширяющую семантику ключевых элементов. В языке UML существует три конкретных механизма расширения, которые включают стереотип (stereotype), помеченное значение (tagged value), ограничение (constraint).

Стереотип (stereotype) – новый элемент модели, расширяющий семантику метамодели. Стереотипы должны основываться на существующих типах или классах и описываются в метамодели UML.

Стереотипы предназначены для расширения семантики, но не таких структур или классов, которые уже описаны. Некоторые стереотипы определены заранее в языке UML, другие могут быть указаны разработчиком. На диаграммах отображаются в виде текста, заключенного в угловые кавычки. Стереотипы по умолчанию — это ключевые слова языка UML, используемые в канонических диаграммах на исходном языке без перевода.

Помеченное значение (tagged value) – конкретное определение свойства в виде пары «имя-значение». При указании значения само имя называется тегом.

Помеченные значения отображаются на диаграммах в виде текстовой строки в специальном формате, заключенной в фигурные скобки. Используется следующий формат ввода: {тег = значение}. Теги включены в запись UML, но их определение не является строгим, поэтому разработчик может определять теги самостоятельно[28].

Ограничение (constraint*)* – это некоторые логические условия, которые ограничивают семантику выбранного элемента модели.

Как правило, все ограничения устанавливаются разработчиком. Ограничения на диаграммах представляются в виде строки текста, заключенной в фигурные скобки. Для формального задания ограничений предусмотрен специальный объектно-ориентированный язык ограничений (OCL), являющийся неотъемлемой частью языка UML.

Большинство перечисленных выше *диаграмм* являются графами специального вида, состоящими из вершин в форме геометрических фигур, которые связаны между собой ребрами или дугами. Поскольку *информация*, которую содержит в себе *граф*, носит топологический характер, ни геометрические размеры, ни расположение элементов *диаграмм* не имеют принципиального значения.

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

Геометрические фигуры на плоскости, выступающие в качестве 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 описанию отдельных элементов языка UML (класс, вариант использования, ситуация, деятельность). Графические примитивы UML имеют фиксированную семантику, которая не переопределяется пользователями. Графические примитивы должны иметь собственные имена и, возможно, другой текст, расположенный внутри соответствующих геометрических фигур или, в качестве исключения, рядом с ними.

Графические отношения, представленные различными линиями на плоскости. Отношения в языке UML обобщают концепцию дуг и ребер из теории графов, но менее формальны и имеют более сложную семантику.

Специальные графические символы, которые изображаются рядом с определенными визуальными элементами диаграмм и имеют характер дополнительной спецификации (украшений).

Все диаграммы описываются на языке UML с помощью фигур на плоскости. Отдельные элементы создаются с использованием геометрических фигур, которые могут иметь различную высоту и ширину, чтобы вместить внутри себя другие конструкции языка UML. Часто внутри таких символов размещаются текстовые строки, которые уточняют семантику или корректируют отдельные свойства соответствующих элементов языка UML. Информация, содержащаяся в фигурах, важна для конкретной модели проектируемой системы, поскольку она регулирует работу соответствующих элементов в программном коде.

Пути представляют собой последовательность отрезков, соединяющих отдельные графические символы. В этом случае точки отрезков должны касаться геометрических фигур, отображающих вершины диаграмм, как это обычно бывает в теории графов. С концептуальной точки зрения, в языке UML путям придается особое значение, поскольку они являются простыми топологическими сущностями. Отдельные части пути или сегменты не могут существовать за пределами пути, которая их содержит. Пути всегда касаются других графических символов на обеих границах соответствующих отрезков, т.е. пути не могут обрываться на диаграмме линией, которая не касается ни одного графического символа. Как упоминалось выше, дороги могут иметь особую графическую форму, например, голову или терминатор - значок, показанный на одном из концов линий.

Дополнительные значки или украшения - это графические изображения фиксированного размера и формы. Они не могут увеличиваться в размере для размещения дополнительных символов. Значки размещаются внутри и снаружи других графических элементов.

§ 2. Диаграмма вариантов использования

§ 2.1 Диаграмма вариантов использования: понятие, задачи и требования

Диаграмма вариантов использования – это концептуальное представление или концептуальная модель системы, созданная в процессе проектирования и разработки. Сценарий использования (UC) – это подробное описание того, как пользователь взаимодействует с системой для достижения конкретной цели. Вариант использования как универсальный инструмент, может быть полезен в любой сфере разработки программного обеспечения (веб-сайт для интернет-магазина, приложение для медицинского персонала т.д.), поможет объяснить требования и пошагово описать поведение пользователя в конкретной ситуации.

Создание диаграммы вариантов использования преследует следующие цели:

– Определение общих границ и контекста моделируемой предметной области на начальных этапах проектирования системы.

– Формулирование общих требований к функциональному поведению проектируемой системы.

– Разработка концептуальной модели системы для ее последующего детального описания в виде логических и физических моделей.

– Подготовка первоначальной документации для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Как и любой другой инструмент, диаграмма вариантов использования имеет свои преимущества и ограничения.

Вариант использования помогает команде и клиентам обсуждать ключевые сценарии использования, чтобы отклонить неэффективные решения или предложить альтернативы в плане изменения процесса.

Прозрачность деловой координации с клиентом. Каждый вариант использования имеет конкретную бизнес-ценность, понятную заинтересованной стороне.

Помогает проверить полноту требований к системе. Использование этого вида диаграммы позволяет выявить пробелы и недостатки в функциональности, которые могут быть упущены при использовании других методов анализа требований предметной области.

Таким образом, на основе диаграмм вариантов использования составляется полная и подробная проектная документация, которую удобно организовать для работы всей команды.

Но существуют определённые ограничения использования диаграмм вариантов использования. Если основную роль в приложении играют не действия пользователя, как в трекерах задач с уникальными жизненными циклами, типами и связями, а внутренние правила взаимодействия объектов, то вариант использования не будет эффективен.

Вариант использования не покрывает нефункциональные требования. Например, скорость системы, безопасность, масштабируемость и т. д.

Вариант использования для крупных проектов может быть очень сложным и трудным для понимания. Если в системе много взаимосвязанных функций, каждый вариант использования может потребовать множества уточнений.

Качество сценариев использования зависит от опыта человека, который их написал. Неопытный аналитик может упустить важные детали или неправильно понять требования, что приведет к некорректным сценариям.

§ 2.2 Отношения на диаграмме вариантов использования

На диаграммах вариантов использования (Use Case diagram) в UML существует четыре основных отношения между элементами: ассоциация, включение, расширение и обобщение, которые описывают взаимодействие акторов и вариантов использования, также связи между самими вариантами использования (рис.2.1).

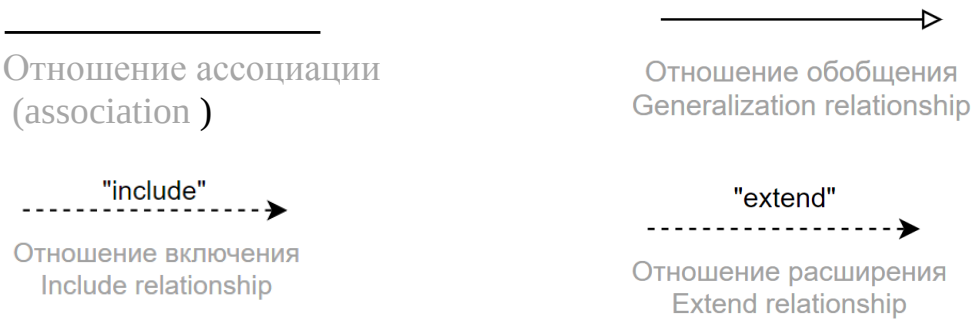


Рисунок 2.1

Ассоциация рассматривается как связь между актором и вариантом использования, которая описывает роль актора при взаимодействии с конкретным вариантом использования. Она указывает на то, что актор может инициировать выполнение варианта использования (главный актор) или получать информацию о результатах его выполнения (второстепенный актор).

Эта связь изображается графически в виде сплошной линии между актором и вариантом использования, но может иметь дополнительные графические сведения, как имя и кратность.

Отношение зависимости между двумя вариантами использования, при котором один вариант (базовый) обязательно включает в себя функциональность другого варианта как составной части называется отношением включения. Это означает, что при каждом выполнении базового вариант использования всегда должен выполняться и включаемый вариант. Графически включение представлено пунктирной линией со стрелкой, которая направлена от базовых вариант использования в включаемому и помечается стереотипом <<include>>.

Отношение расширения определяет взаимосвязь базового варианта использования с другим вариантом, который реализуется не всегда, а только при выполнении определённых условий. Графически расширение обозначается пунктирной линией со стрелкой, направленной от расширяющего варианта использования к базовому. Пунктирная линия со стрелкой помечается стереотипом <<extend>>.

Отношение обобщения показывает, что один элемент модели (дочерний) является специальным или частным случаем другого родительского элемента. Дочерний элемент наследует все атрибуты, операции и отношения родительского элемента. В контексте диаграмм вариантов использования обобщение может применяться как между акторами, так и между вариантами использования. Графически обобщение обозначается сплошной линией со стрелкой в форме не закрашенного треугольника, которая направлена от дочернего (частного) элемента к родительскому.

Сравнительная характеристика отношений на диаграмме использования представлена в таблице № 1.

Таблица № 1 – Сравнительная характеристика отношений на диаграмме использования

Название	Использование	Графическое представление
Ассоциация	Связь актора с вариантом использования	Сплошная линия
Обобщение	Иерархическая связь «общее-частное»	Сплошная линия со стрелкой в форме не закрашенного треугольника
Включение	Обязательное включение одного варианта в другой	Пунктирная линия со стрелкой и стереотипом<<include>>
Расширение	Условное добавление поведения к базовому варианту	Пунктирная линия со стрелкой и стереотипом<<extend>>

Эти отношения помогают моделировать зависимости и взаимодействия в системе, сокращая дублирование и повышая читаемость диаграмм.

§ 2.3 Дополнительные обозначения языка UML для бизнес-моделирования

Язык UML включает в себя специальные механизмы расширения, позволяющие указывать дополнительные графические символы, предназначенные для решения задач из конкретной предметной области. Примеры подобных записей, используемых для моделирования бизнес-систем и описываемых в диаграммах вариантов использования: бизнес-актор, сотрудник и бизнес-вариант использования.

Бизнес-актор – это отдельный человек, группа, организация, компания или система, взаимодействующие с моделируемой бизнес-системой, но не являющиеся её частью, то есть не являющиеся частью моделируемой системы.

Графическое представление бизнес-актера показано на рис. 2.2, а. Примерами бизнес-субъектов являются клиенты, покупатели, поставщики, партнеры. Общим свойством бизнес-субъектов является то, что они являются инициаторами или клиентами бизнес-процессов моделируемой системы.

Сотрудник – это человек, который работает в моделируемой бизнес-системе, взаимодействует с другими работниками и участвует в бизнес-процессе моделируемой системы.

Бизнес-вариант использования – это вариант использования, определяющий последовательность действий моделируемой системы для реализации отдельного бизнес-процесса.

Графическое представление бизнес-сценария показано на рисунке 2.2, в. Общей чертой бизнес-сценариев является то, что моделируемая система представляет собой концептуальную модель отдельных бизнес-процессов.

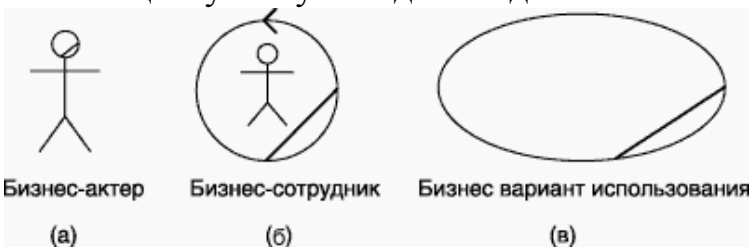


Рисунок 2.2 - Графические изображения бизнес-актера (а), бизнес-сотрудника (б) и бизнес-варианта использования (в)

§ 2.4 Рекомендации по разработке диаграмм вариантов использования

Как уже упоминалось, одной из главных целей диаграммы вариантов использования является формализация функциональных требований к системе. Диаграмма вариантов использования может служить основой для согласования функциональных требований к системе с заказчиком на ранних этапах проектирования. Любой из них в дальнейшем может быть декомпозирован на другие варианты использования. При этом общее количество акторов не должно быть больше 20, а количество вариантов использования не больше 50.

При разработке диаграмм вариантов использования рекомендуется придерживаться следующих правил.

1. Определение основных первичных и второстепенных акторов.
2. Определение целей основных акторов по отношении к систем.
3. Формулирование основных вариантов использования, определяющих функциональные требования к системе.
4. Ранжирование вариантов использования в соответствии со степенью снижения риска их реализации.
5. Выделение участников, интересов, предварительных условий и последующих событий для выбранного варианта использования.
6. Написание успешного сценария реализации выбранного варианта использования.
7. Определение исключений или неудач при выполнении сценария варианта использования.
8. Написание сценария для всех исключений.
9. Выделение основных вариантов использования и описать их взаимосвязи, используя стереотип <<include>>.
10. Выделение основных вариантов использования и описать их взаимосвязи, используя стереотип <<extend>>.
11. Проверить диаграмму, чтобы избежать дублирования вариантов использования и акторов.

12. При построении диаграммы часто возникает путаница относительно направления стрелки. Следует помнить одно правило: стрелка обычно направлена от «зависимого» объекта к «независимому» объекту.

13. Не нужно засорять диаграмму слишком мелкими действиями. Объедините все общие действия в одну группу под общим названием, чтобы было просто читать диаграмму.

14. Старайтесь не допускать пересечений соединительных линий. Это может затруднить чтение диаграммы для вас и для ваших коллег.

15. Не дублируйте варианты использования на диаграмме. Если приходится дублировать варианты использования, то элементы диаграммы надо постараться расставить по-другому.

16. Пользуйтесь специальными компьютерными программами для построения диаграмм. Это существенно упростит весь процесс моделирования.

17. Диаграммы очень легко изменить. Не нужно бояться, что требования к программному обеспечению изменятся или вы забудете что-то показать на диаграмме. Можно добавлять элементы на диаграмму в любое время.

18. Диаграмма не должна быть перегружена слишком мелкими движениями. Можно объединить все общие действия в группу с общим названием, чтобы диаграмма была легко читаемой.

19. Необходимо избегать пересечения соединительных линий. Это может затруднить чтение графика.

20. Не повторять варианты использования на диаграмме. Если необходимо повторить варианты использования, надо попытаться расположить элементы диаграммы по-другому.

§ 2.5 Создание нового проекта в StarUML

Основная структурная единица в StarUML – это проект. Он может содержать модель и различные представления этих моделей. Новый проект будет автоматически создан при запуске программы StarUML. В верхней части основного окна (рисунок 1) расположено главное меню. Слева расположены список рабочих диаграмм и панель инструментов с изображениями элементов диаграммы. Элементы соответствуют типу выбранной диаграммы. В центре находится рабочее поле диаграммы, на котором она может быть построена с использованием соответствующих элементов панели инструментов. Справа находятся навигатор Explorer и редактор свойств Editor, состоящий из редактора стилей Styles, редактора свойств и окна документирования элементов модели Documentation. Внешний вид редактора представлен на рисунке 2. 3

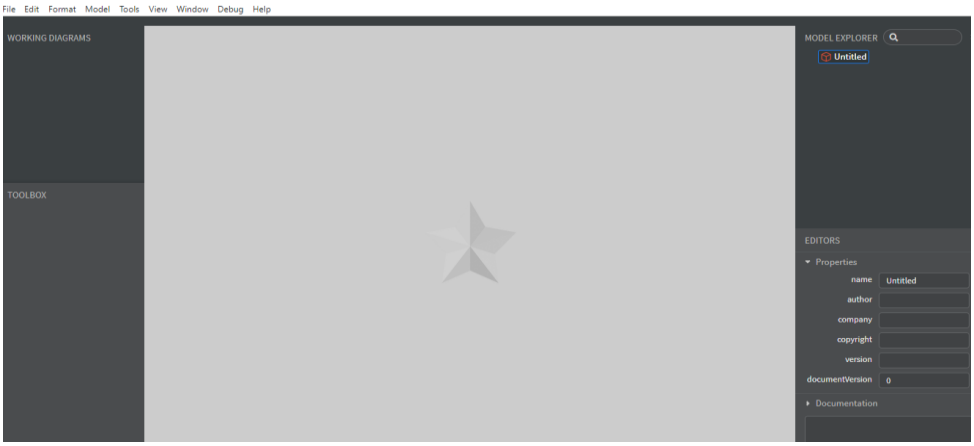


Рисунок 2.3 – Основное окно программы

При нажатии File>New From Template> Rational. Слева, в навигаторе отображаются виды диаграмм (рис.2.4)

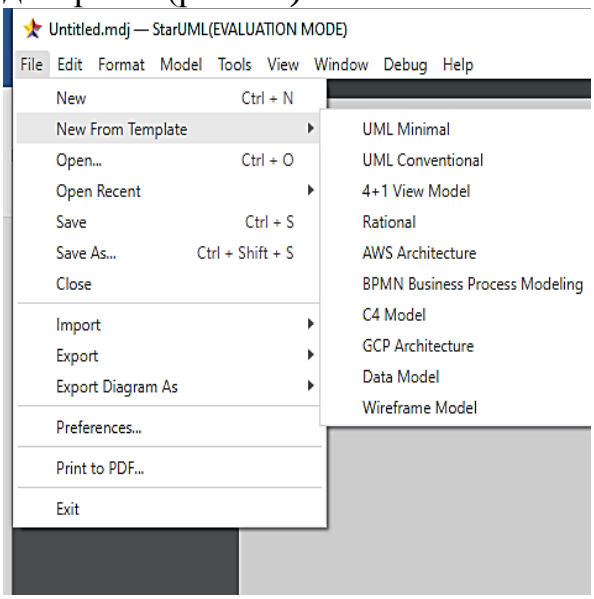


Рисунок 2.4

Управлять видом инспектора модели, панели элементов, закрывать и открывать редакторы инспектора можно с помощью пункта меню View (рисунок 2.5).

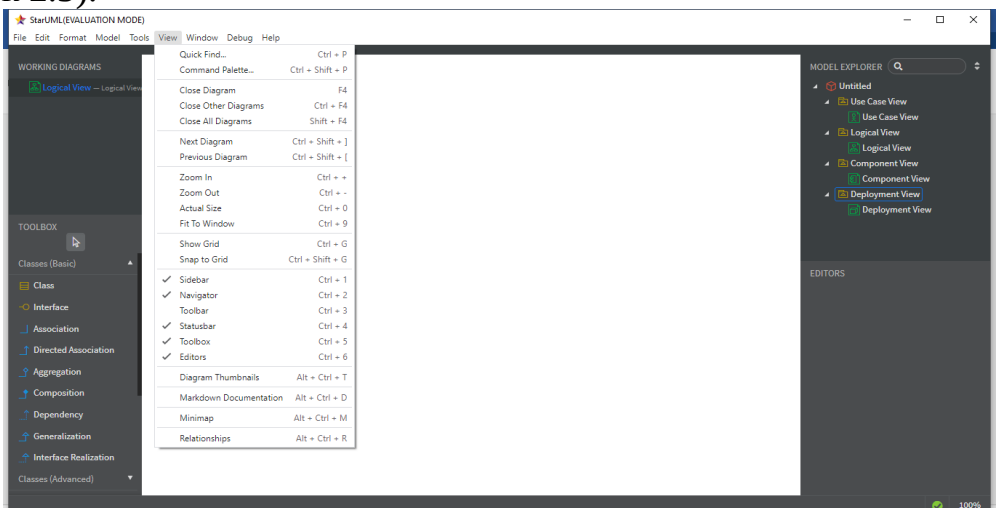


Рисунок 2.5 – Пункт меню View

Иерархическая структура проекта отображается справа в навигаторе Model Explorer (рисунок 2.6). В зависимости от выбранного подхода на навигаторе модели будут отображены различные представления модели. Каждое представление будет содержать элементы моделей и диаграмм, которые надо создать.

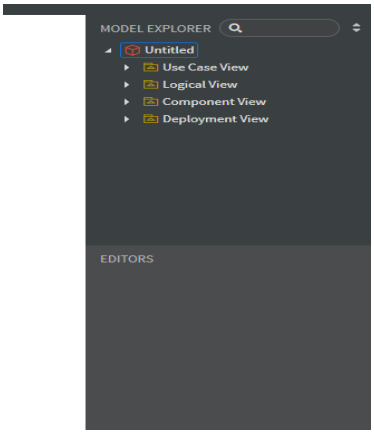


Рисунок 2.6 – Навигатор

Для того, чтобы добавить новую диаграмму, нужно выбрать в навигаторе модель, нажать на ней правой кнопкой и в выпадающем меню раскрыть список Add Diagram (рис. 2.7).

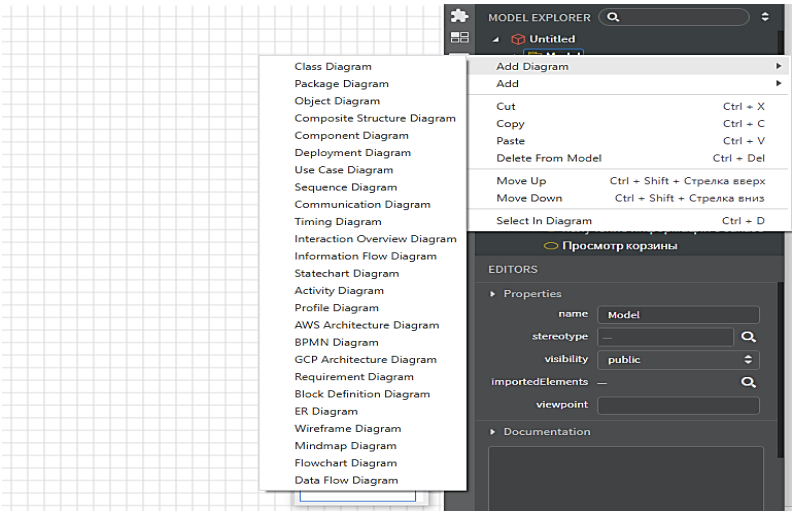


Рисунок 2.7 – Добавление диаграммы

Рассмотрим визуальное моделирование систем с использованием UML на примере создания диаграммы вариантов использования при проектировании системы заказов интернет-магазина. С помощью диаграммы вариантов использования можно:

1. Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;

2. Сформулировать общие требования к функциональному поведению проектируемой системы;

3. Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;

4. Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Диаграмма вариантов использования (прецедентов) представляет собой граф, в вершинах которого расположены актеры или прецеденты, связи между вершинами – это разного вида отношения. Актером (действующее лицо, actor) называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне.

Это может быть человек, техническое устройство, программа или любая другая система, которая служит источником воздействия на моделируемую систему так, как определит разработчик. На диаграммах вариантов использования актер изображается в виде человечка, под которым записывается его имя.

Прецедент описывает, что делает система, но никак не уточняет, как она это делает. Заметим, что диаграмма прецедентов не отображает последовательность, в которой будут выполняться варианты использования. На диаграмме прецедент изображается в виде эллипса. Имя прецедента может состоять из нескольких слов и знаков препинания (за исключением двоеточия), как правило, имя выбирают в виде словосочетания или глагольного выражения (рис. 2.8).

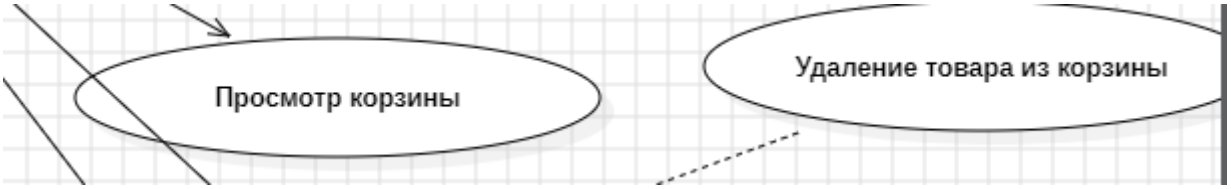


Рисунок 2.8 – Варианты использования (прецеденты)

Необходимо разработать модель прецедентов (вариантов) использования, состоящую из одной или нескольких диаграмм прецедентов использования, для разрабатываемой информационной системы.

Постановка задачи (описание предметной области). Магазин осуществляет продажу товаров клиенту путем оформления документов «Заказ». Директор магазина, принял решение автоматизировать документооборот продаж товара и пригласил для выполнения работ программиста. Поговорив с директором, в соответствие с концепцией жизненного цикла (ЖЦ) программы программист приступил к описанию бизнес процессов, сопровождающих продажу товара. Взяв за основу язык UML, он начал с построения контекстной диаграммы процессов-Use Case diagram. Диаграмма должна ответить на вопрос «Что должно делаться в системе, и кто участник этих процессов?». Окончательный вид диаграммы показан на рис.2.9

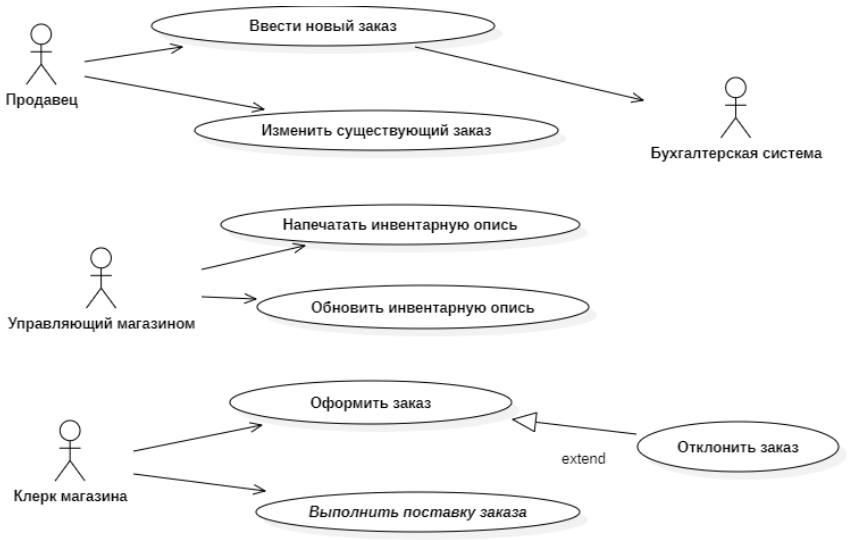


Рисунок 2.9 – Диаграмма вариантов использования задачи о заказе товара

Для этого необходимо выполнить следующую последовательность действий:

1. Создать новый проект и заполнить его свойства в области EDITORS.
2. В навигаторе моделей выбрать <<UseCaseModel>>, и с помощью контекстного меню выбрать действие «AddDiagram» и создать диаграмму прецедентов использования (UseCaseDiagram) для отображения границ системы. Присвоить диаграмме наименование (область Properties) (рис.3.0).

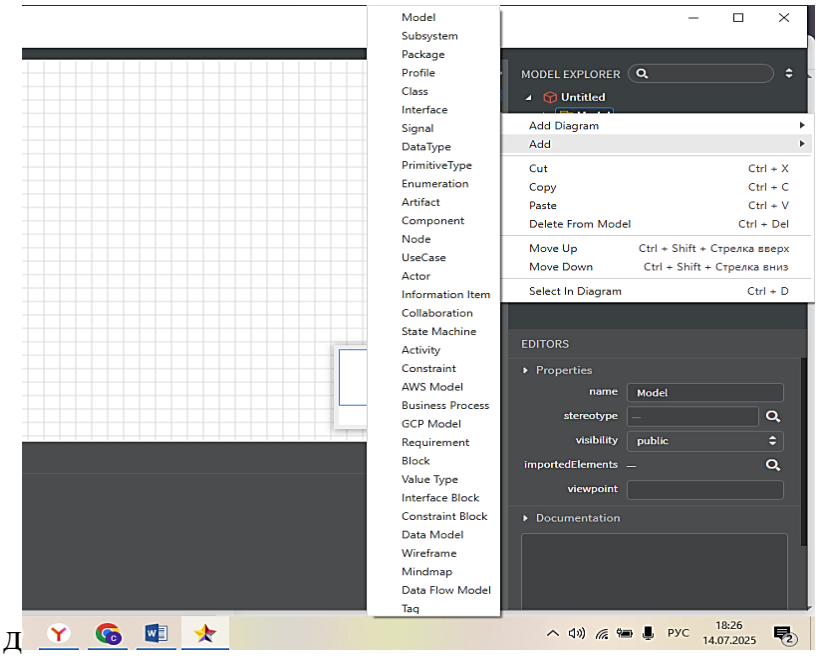


Рисунок 3.0 – Создание диаграммы прецедентов использования

3. С помощью окна «Toolbox» и вкладки «UseCase» поместить на диаграмму новый вариант использования. Назвать его «Ввести новый заказ».

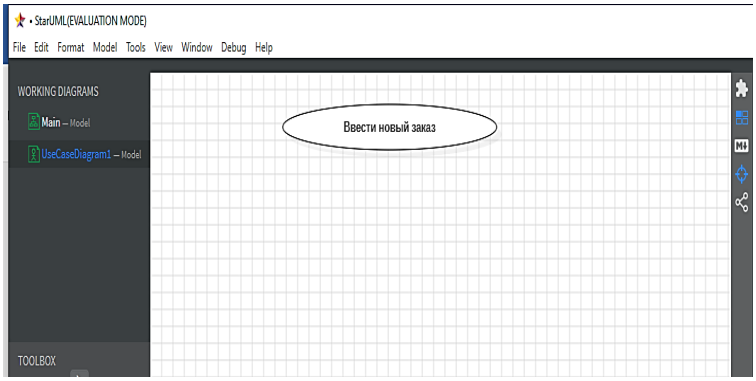


Рисунок 3.1 – Создание диаграммы прецедентов использования

4. Повторив этапы 2 и 3, поместить на диаграмму остальные варианты использования:

– Изменить существующий заказ;
– Напечатать инвентарную опись;
– Обновить инвентарную опись;
– Оформить заказ;
– Отклонить заказ;
– Выполнить поставку заказа.

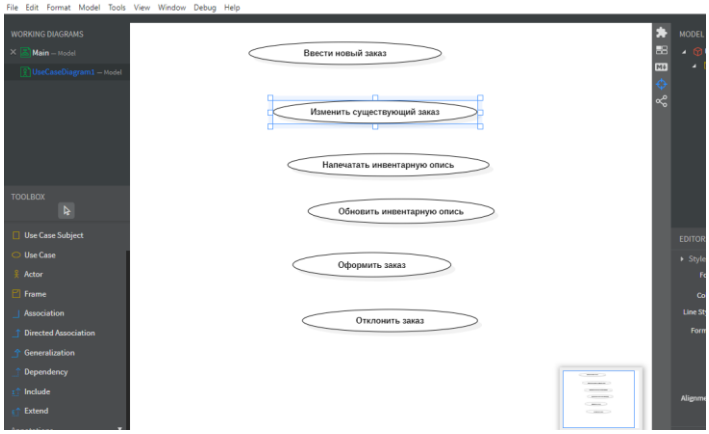


Рисунок 3.2 – Создание диаграммы прецедентов использования

5. С помощью кнопки Actor (Действующее лицо) панели инструментов поместить на диаграмму новое действующее лицо.

6. Назвать его «Продавец».

7. Повторив шаги 4 и 5, поместить на диаграмму остальных действующих лиц:

– управляющий магазином;
– клерк магазина;
– бухгалтерская система.

8. Создание абстрактного варианта использования (не требующего дальнейшей декомпозиции).

Щелкнуть мышью на варианте использования «Отклонить заказ» на диаграмме.

В окне «EDITORS» установить флажок isAbstract (Абстрактный), чтобы сделать этот вариант использования абстрактным (рис.3.4).

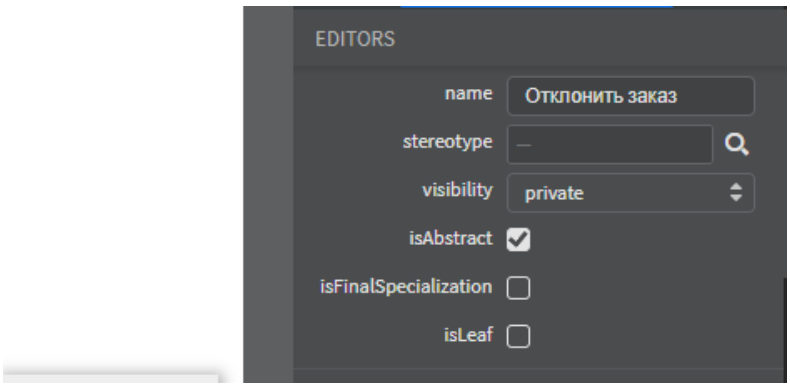


Рисунок 3.4 – Создание абстрактного варианта использования

9. Добавление ассоциаций. С помощью кнопки Directed Association (Однонаправленная ассоциация) в окне «Toolbox» нарисовать ассоциацию между действующим лицом Продавец и вариантом использования «Ввести заказ». Повторив шаг 1, поместить на диаграмму остальные ассоциации, согласно рис.8.

10. Добавление связи расширения. С помощью кнопки Generalization(Обобщение) в окне «Toolbox» нарисовать связь между вариантом использования «Отклонить заказ» и вариантом использования «Оформить заказ». Стрелка должна быть направлена от первого варианта использования ко второму. Связь расширения означает, что вариант использования «Отклонить заказ» при необходимости дополняет функциональные возможности варианта использования «Оформить заказ».

11. Добавление описаний к вариантам использования. Выделить вариант использования «Ввести новый заказ». В окне документации ввести следующее описание: «Этот вариант использования дает клиенту возможность ввести новый заказ в систему» (рис.3.5).

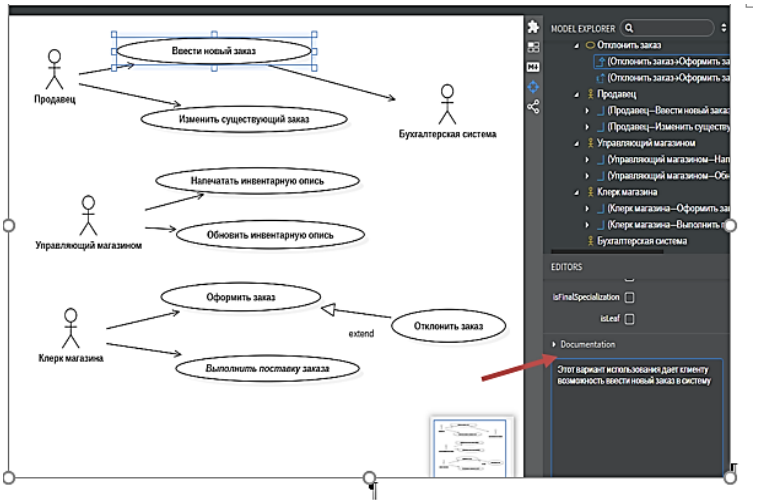


Рисунок 3.5 – Описание к вариантам использования

С помощью окна документации добавить описания ко всем остальным вариантам использования.

12. Добавление описаний к действующему лицу. Выделить действующее лицо «Продавец». В окне документации ввести следующее

описание: «Продавец – это служащий, старающийся продать товар». С помощью окна документации добавить описания к остальным действующим лицам.

Структура спецификации прецедента использования (Use Case Document) может варьироваться, однако типичное описание должно содержать следующую информацию: краткое описание, участвующие действующие лица, предусловия, необходимые для инициирования прецедента использования.

Детализированное описание потока событий, включающее в себя:

– основной поток событий, который можно разбить на несколько частей, чтобы показать подчиненные потоки событий (подчиненные потоки могут быть разделены затем на еще более мелкие потоки, с целью улучшить читабельность документа);

– альтернативные потоки для определения исключительных ситуаций;

– постусловия, определяющие состояние систем по достижении, которого прецедент использования завершается.

Вопросы для самоконтроля

1. Какие элементы может содержать диаграмма вариантов использования?
2. Чем или кем могут быть представлены акторы в системе?
3. В каких случаях используют отношения включення на диаграммах вариантов использования?
4. В каких случаях используют отношения расширения на диаграммах вариантов использования?
5. Как определяется направленность связи ассоциации между актером и вариантом использования?
6. Что такое точка расширения прецедента?
7. С помощью чего описывается динамика вариантов использования.
8. На каком этапе разработки программной системы разрабатывается диаграмма вариантов использования? Каково ее назначение?
9. Сколько диаграмм вариантов использования необходимо разрабатывать для одной программной системы?

§ 3. Диаграммы взаимодействия (Interaction diagrams)

§ 3.1 Диаграмма последовательности (Sequence-диаграммы)

Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов. Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми ни обмениваются между собой.

Сообщение (message) – это средство, с помощью которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций.

Информационное (informative) сообщение – это сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) – это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

Императивное (imperative) сообщение – это сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует два вида диаграмм взаимодействия: диаграммы последовательности (sequencediagrams) и кооперативные диаграммы (collaborationdiagrams).

Диаграмма последовательности показывает поток событий, происходящих в вариантах использования.

Диаграммы последовательности отображают взаимодействие объектов во времени. Они помогают понять сложные процессы, выявить узкие места и потенциальные проблемы на ранней стадии разработки, а также сэкономить время и ресурсы благодаря следующим особенностям: четкие требования, мост между аналитиком и разработчиком, рабочая документация, упрощение тестирования,

Графическое описание помогает более точно сформулировать требования, снижая вероятность недопонимания и последующих изменений. Это особенно ценно при работе с распределенными командами или, когда в проекте участвуют специалисты с разным техническим опытом.

Диаграммы как инструмент коммуникации служат для того, чтобы обе стороны могли обсуждать процесс продуктивно. Благодаря этим диаграммам сокращается количество звонков и переписки для уточнения требований.

Диаграммы последовательности четко и ясно отражают процессы и компоненты логического взаимодействия. При этом вместо текстовых описаний используются графические изменения, которые удобно использовать.

Для тестировщиков диаграммы являются отличной основой для разработки тестовых сценариев, обеспечивая более полное покрытие и выявление потенциальных ошибок.

Хотя диаграммы последовательности широко используются в системном анализе, это не единственный способ описания взаимодействий в системе. Важно понимать, что они лучше или хуже других методов в зависимости от ситуации.

Диаграммы активности (Activity diagrams) подходят для описания общего потока действий, особенно если уделяется внимание действиям и логике принятия решений, но они плохо передают взаимодействие между участниками.

Диаграммы состояний (State diagrams) подходят для отображения жизненного цикла объекта, но они не показывают взаимодействия между объектами.

BPMN (Business Process Model And Notation) подходит для бизнес-аналитиков, показывает процессы с акцентом на роли, события, потоки, но не отражают последовательность вызовов между системами и компонентами.

Диаграммы последовательности фокусируются на временной оси и конкретных взаимодействиях между участниками.

Общие рекомендации для создания диаграмм последовательности

1. Четко определите, кто с кем взаимодействует и в какой момент, зафиксируйте порядок вызовов и ответов.

2. Отразите сложные сценарии с ветвлениями, параллельными процессами и обработкой ошибок в контексте реального времени.

3. Отобразите взаимодействия всех участников одновременно, включая интерфейсы, бэкэнд, внешние сервисы и точки интеграции, важные для сложных распределенных систем.

4. Сформируйте общее видение для разных ролей: системных аналитиков, разработчиков и тестировщиков.

Sequence-диаграммы не заменяют другие инструменты, но в ситуациях, когда важна последовательность событий и точность коммуникации между компонентами, они являются более предпочтительным вариантом.

Диаграмма может показать, как мобильное приложение взаимодействует с платежной системой через REST API (Representational State Transfer), обрабатывая запросы, ответы и ошибки для всех участников

Для отображения рабочих процессов можно использовать диаграммы последовательности различных взаимодействий между участниками. Диаграмма может показывать процесс обработки заказа в интернет-магазине, демонстрируя взаимодействие между клиентом, системой обработки заказов, складом и службой доставки.

Диаграммы последовательности наглядно показывают, как различные компоненты системы (объекты, классы, подсистемы) взаимодействуют во времени. Диаграмма может показать, как запрос баланса счета пользователя проходит через сервер, службу обработки платежей и базу данных, демонстрируя передачу данных и вызовы методов на каждом этапе.

Перед началом построения последовательности диаграмм для UML (унифицированного языка моделирования) необходимо знать основные элементы (рис.3.1). Рассмотрим их подробнее:

Участники (Объекты). Последовательность диаграмм в модели взаимодействия системы для различных типов участников включает:

Акторы – это внешние сущности, взаимодействующие с системой, такие как пользователи или другие системы. Системы также начинают отправлять запросы и получают ответы.

Границы – это точки, определяющие границы взаимодействия системы с внешним миром, такие как пользовательский интерфейс или API.

Контроллеры – контролируют поток данных и логику системы, обрабатывают запросы от участников и перенаправляют их соответствующим сущностям.

Сущности – часто представляют данные и состояние системы, такие как базы данных или другие хранилища данных.



Рисунок 3.1

На диаграмме последовательности каждый объект представлен вертикальной временной шкалой, показывающей период существования и активности. Взаимодействие между объектами обозначается сообщениями (стрелками), которые могут передавать информацию и сведения о состоянии объекта.

Вертикальная пунктирная линия, отражающая присутствие и участие субъекта во взаимодействии называется линией жизни. Она начинается с создания объекта и заканчивается удалением или завершением взаимодействия. На линии жизни могут отображаться действия, ситуации и временные метки для объяснения последовательности событий.

Горизонтальные стрелки, показывающие взаимодействие между участниками – это сообщения. Они показывают передачу данных или вызов метода.

Сообщения могут быть разных типов:

Синхронные сообщения: сплошная стрелка с закрашенным наконечником указывает на то, что отправитель ожидает ответа от получателя для продолжения работы. Например, вызов функции, которая блокирует свое выполнение до получения результата.

Асинхронные сообщения: сплошная стрелка с не закрашенным наконечником указывает на то, что отправитель не ожидает немедленного ответа. Например, отправка сообщения в очередь или приложение, работающее в фоновом режиме (рис.3.2).

Возвратные сообщения: пунктирная стрелка с не закрашенным наконечником, идущая от получателя к отправителю, указывает на возвращаемое значение функции или метода.

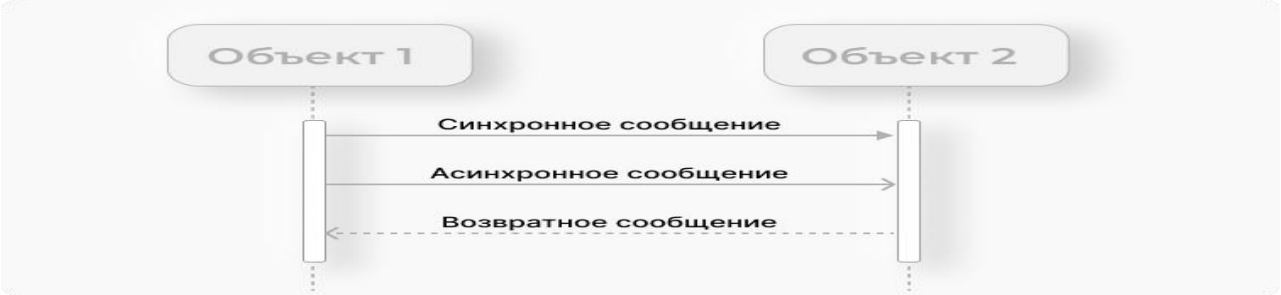


Рисунок 3.2

Активация – прямоугольники, расположенные на линиях жизни, которые отражают период активности объекта. Они указывают на то, когда объект выполняет некоторую операцию или метод (рис.3.3).



Рисунок 3.3

Примечания – используются для добавления пояснений или дополнительной информации к диаграмме. Они помогают улучшить читабельность и понятность. Обозначаются прямоугольником с загнутым краем, соединенным пунктирной линией с соответствующим элементом диаграммы (рис.3.4).



Рисунок 3.4

При создании диаграмм последовательности можно использовать фреймы как специальные блоки, позволяющие создавать сложные взаимодействия, такие как циклы, альтернативные варианты действий или параллельные процессы. Рассмотрим примеры фреймов и их обозначения.

– alt (условное действие) – указывает на логику ветвления в зависимости от условия. Состоит из двух или более частей, одна из которых активируется в зависимости от условий

– opt (опциональное действие) – указывает на блок, который может быть выполнен или пропущен в зависимости от условия.

– loop (цикл) – указывает на блок повторяющихся действий. Включает состояние выхода из цикла.

– par (параллельная операция) – указывает на параллельную работу нескольких блоков.

– ref – ссылается на другую диаграмму последовательности. Используется для упрощения сложных диаграмм путем их разбиения на более мелкие, управляемые части. Это предотвращает чрезмерное усложнение основной диаграммы.

– neg – означает альтернативный сценарий, который не следует реализовывать. Обычно используется для моделирования ошибок или исключительных ситуаций (рис.3.5).

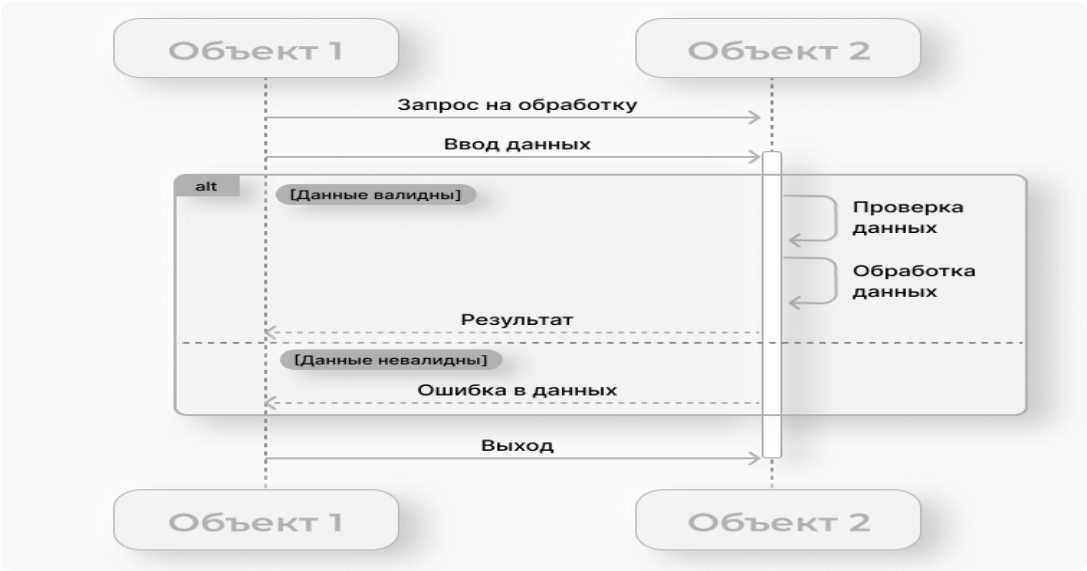


Рисунок 3.5

Уничтожение объекта – элемент обозначающий конец его линии жизни в этом взаимодействии, т.е. момент, когда объект перестает иметь значение в последовательности (рис.3.6).

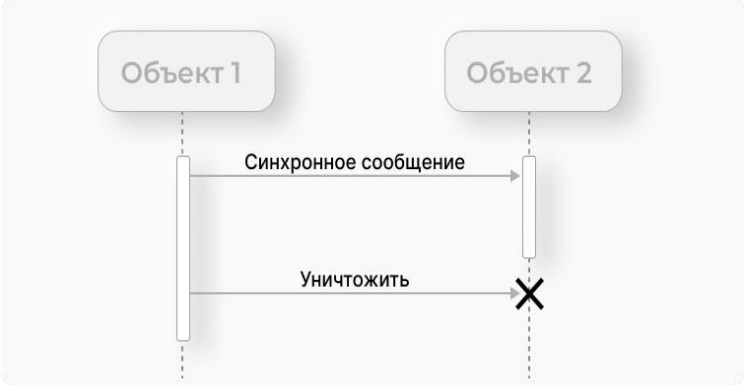


Рисунок 3.6

§ 3.2 Создание Sequence-диаграммы

Постановка задачи. Для варианта использования «Ввод нового заказа» построить диаграммы взаимодействия.

Диаграммы взаимодействия включают в себя два типа диаграмм-Последовательности и Кооперативную

В правом окне программы «ModelExplorer» на вкладке «Model» щелкнув правой кнопкой мыши, выбрать действие «AddDiagram» и выбрать тип диаграммы «SequenceDiagram».

1. Назвать новую диаграмму «Ввод заказа» во вкладке «Properties» в поле «Name».

2. Дважды щелкнув на этой диаграмме, открыть ее.

4.**Добавление на диаграмму действующего лица и объектов.** Перетащить действующее лицо «*Продавец*» из окна «ModelExplorer» из диаграммы использования на диаграмму «Ввод заказа»

5. Выбрать объект «Lifeline» и поместите его на диаграмму.

6. Назвать ее объект *Выбор варианта заказа* (рис.3.7).

7. Повторив шаги 5 и 6, поместить на диаграмму объекты:

– *Форма деталей заказа;*

– *Заказ №1234.*

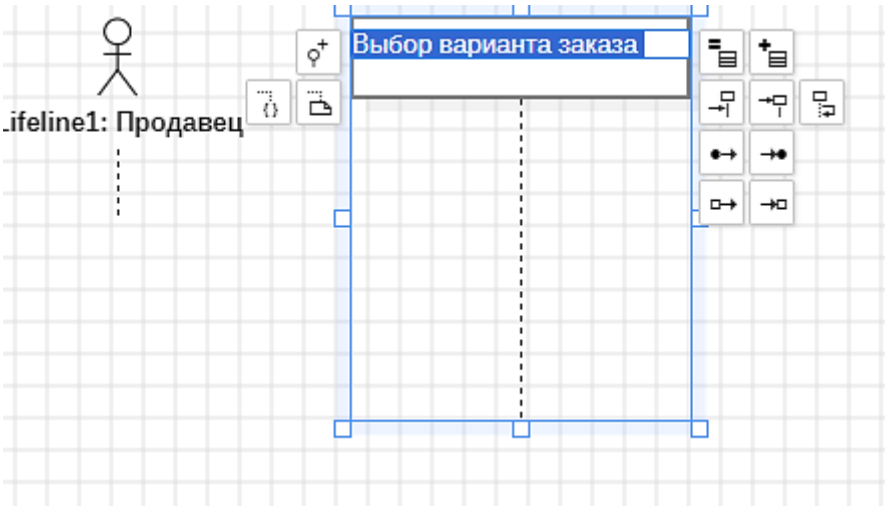


Рисунок 3.7 – Добавление объекта «*Выбор варианта заказа*»

8. **Добавление сообщений на диаграмму.** В окне «Toolbox» нажать кнопку «Message» (Сообщение объекта).

9. Провести мышью от линии жизни действующего лица «*Продавец*» к линии жизни объекта «*Выбор варианта заказа*».

10. Ввести его имя *Создать новый заказ*.

11. Повторив шаги 8 и 9, поместить на диаграмму сообщения:

– *Открыть форму* – между *Выбор Варианта Заказа* и *Форма деталей Заказа;*

– *Ввести номер заказа, заказчика и число заказываемых предметов* – между *Продавец* и *Форма Деталей Заказа;*

– *Сохранить заказ* – между *Продавец* и *Форма Деталей Заказа;*

– *Создать пустой заказ* – между *Форма Деталей Заказа* и *Заказ N1234;*

– *Ввести номер заказа, заказчика и число заказываемых предметов* – между *Форма Деталей Заказа* и *Заказ N1234;*

– *Сохранить заказ* – между *Форма Деталей Заказа* и *Заказ N1234.*

Завершен первый этап работы. Готовая диаграмма Последовательности представлена на рис. 3.8.

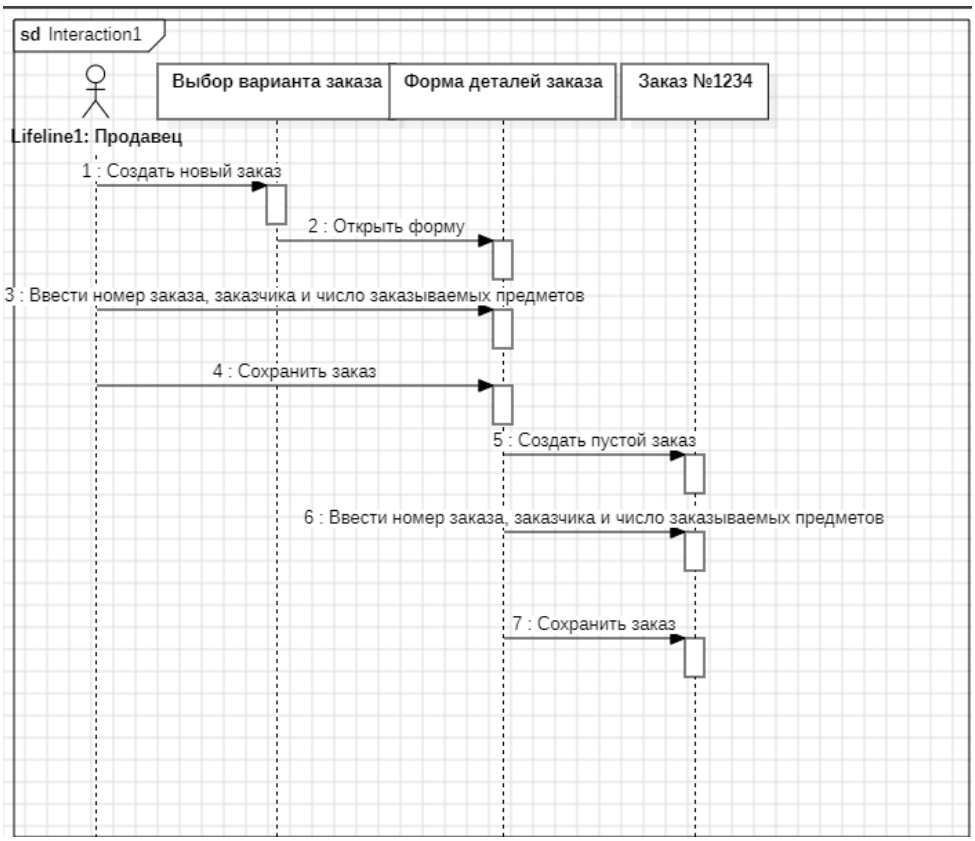


Рисунок 3.8 – Диаграмма последовательности без управляющих элементов

Теперь нужно позаботиться об управляющих объектах и о взаимодействии с базой данных. Как видно из диаграммы, объект *Форма Деталей Заказа* имеет множество ответственностей, с которыми лучше всего мог бы справиться управляющий объект. Кроме того, новый заказ должен сохранять себя в базе данных сам. Вероятно, эту обязанность лучше было бы переложить на другой объект.

12. **Добавление на диаграмму дополнительных объектов.** Поместить новый объект между объектами *Форма Деталей Заказа* и *Заказ №1234*. Ввести имя объекта «*Управляющий заказами»*.

13. Справа от *Заказ №1234* расположить другой новый объект под названием «*Управляющий транзакциями»*.

14. **Назначение ответственностей объектам.** Выделить сообщение №5«*Создать пустой заказ»*.

15. Нажав правой кнопкой мыши на сообщении, удалить его, применив действие «Deletefrom Model».

16. Повторить шаги 14 и 15 для удаления двух последних сообщений:

– *Вести номер заказа, заказчика и число заказываемых предметов;*

– *Сохранить заказ.*

17. Поместить на диаграмму новое сообщение «*Сохранить заказ»*, расположив его под сообщением 4 между *Форма деталей заказа* и *Управляющий заказами*.

18. Повторить шаги 16-17, добавив сообщения с шестого по девятое и назвав их:

Создать новый заказ между–*Управляющий заказами* и *Заказ №1234; Ввести номер заказа, заказчика и число заказываемых предметов* –между *Управляющий заказами* и *Заказ №1234; Сохранить заказ* –между *Управляющий заказами* и *Управляющий транзакциями; Информация о заказе* –между *Управляющий транзакциями* и *Заказ №1234.*

19. В окне «Toolbox» нажать кнопку SelfMessage (Сообщение себе).

20. Щелкнуть на линии жизни объекта *Управляющий транзакциями* ниже сообщения 9, добавив туда рефлексивное сообщение, назвав его «*Сохранить информацию о заказе в базе данных»*.

21. **Соотнесение объектов с классами**.

1. Щелкнуть правой кнопкой мыши на объекте *Выбор варианта заказа*.

2. В открывшемся меню выбрать пункт Create type (создать тип).

3. В раскрывающемся окне в поле Enter an operation name to create ввести *Выбор заказа*.

4. Щелкнуть на кнопке ОК.(рис.3.9-4.0)

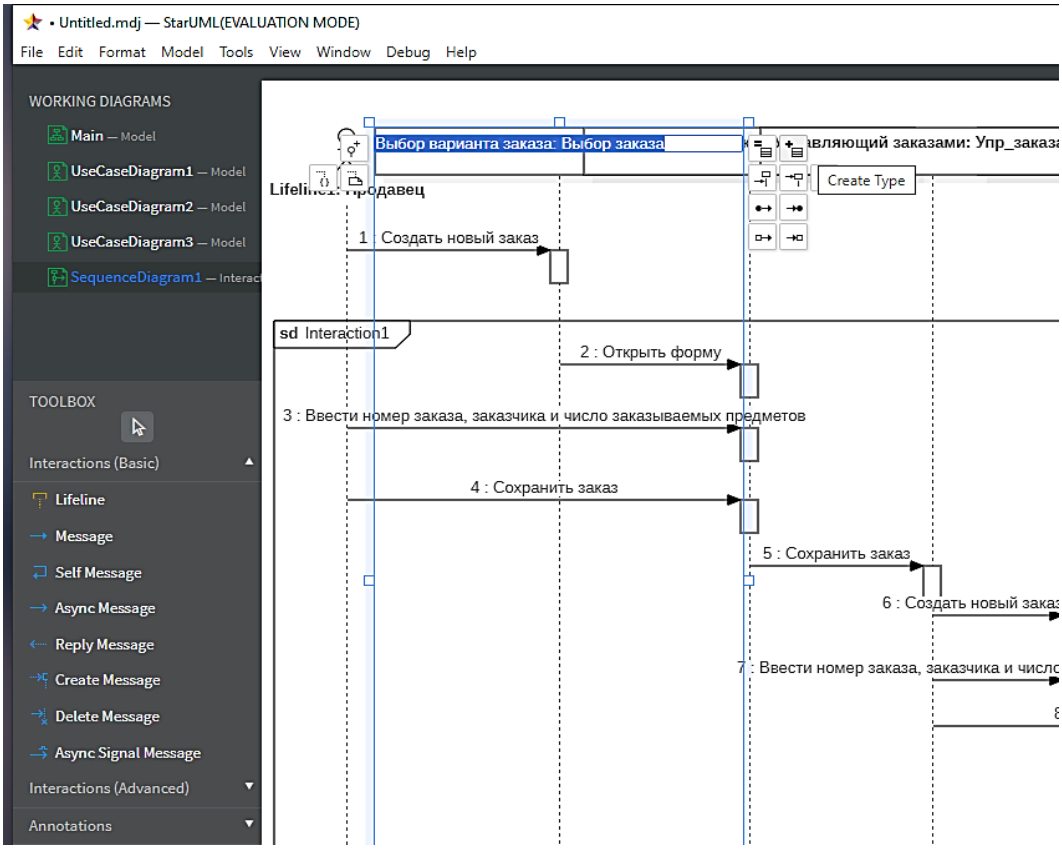


Рисунок 3.9

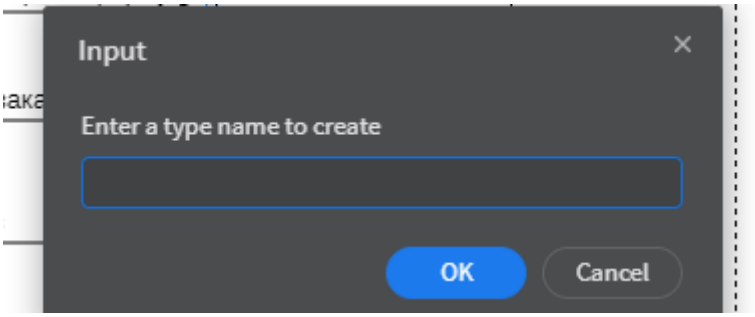


Рисунок 4.0

5. Для соотнесения остальных объектов с классами повторить шаги с 1 по 4:

– Класс *Детали заказа* соотнесите с объектом *Форма деталей заказа*
– Класс *Упр_заказами* – с объектом *Управляющий заказами*
– Класс *Заказ* – с объектом *ЗаказN1234*
– Класс *Упр_транзакциями* – с объектом *Управляющий транзакциями*

Соотнесение сообщений с операциями

1. Щелкнуть правой кнопкой мыши на сообщении 1: *Создать новый заказ.*

2. В открывшемся меню выбрать пункт Create operation (создать операцию).

3. В поле Enter an operation name to create ввести имя операции - *Создать*. Нажмите на кнопку ОК.(рис.4.1-4.2)

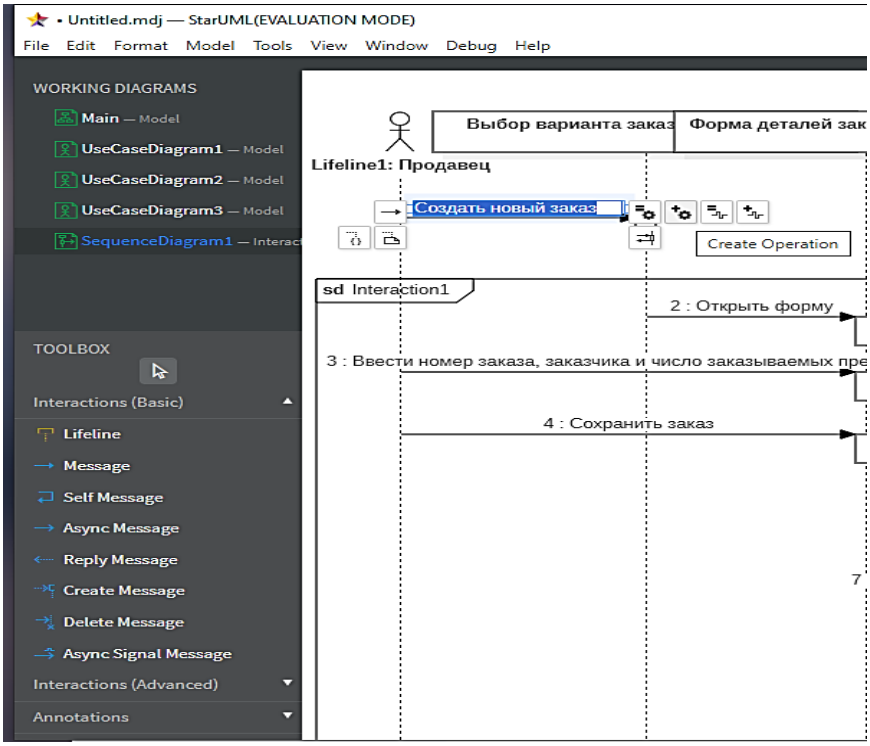


Рисунок 4.1

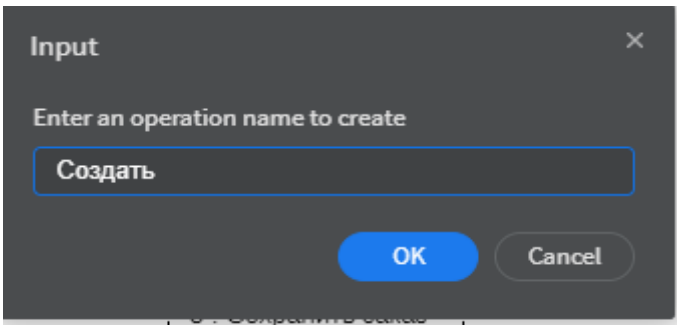


Рисунок 4.2

4. Повторить шаги с 1 по 3, чтобы соотнести с операциями все остальные сообщения:

– Сообщение 2: *Открыть форму* соотнесите с операцией *Открыть()*

– Сообщение 3: *Ввести номер заказа, заказчика и число заказываемых предметов* – с операцией *Ввести номер заказа, заказчика и число заказываемых предметов()*

– Сообщение 4: *Сохранить заказ* – с операцией *Сохранить заказ()*

– Сообщение 5: *Сохранить заказ* – с операцией *Сохранить заказ()*

– Сообщение 6: *Создать пустой заказ* – с операцией *Создать пустой заказ()*

– Сообщение 7: *Ввести номер заказа, заказчика и число заказываемых предметов*-с одноименной операцией.

– Сообщение 8*Сохранить заказ* – с операцией *Сохранить заказ()*

– Сообщение 9 *Информация о заказе* – с одноименной операцией

– Сообщение 10 *Сохранить информацию о заказе* с одноименной операцией. Диаграмма должна выглядеть, как на рис. 4.3.

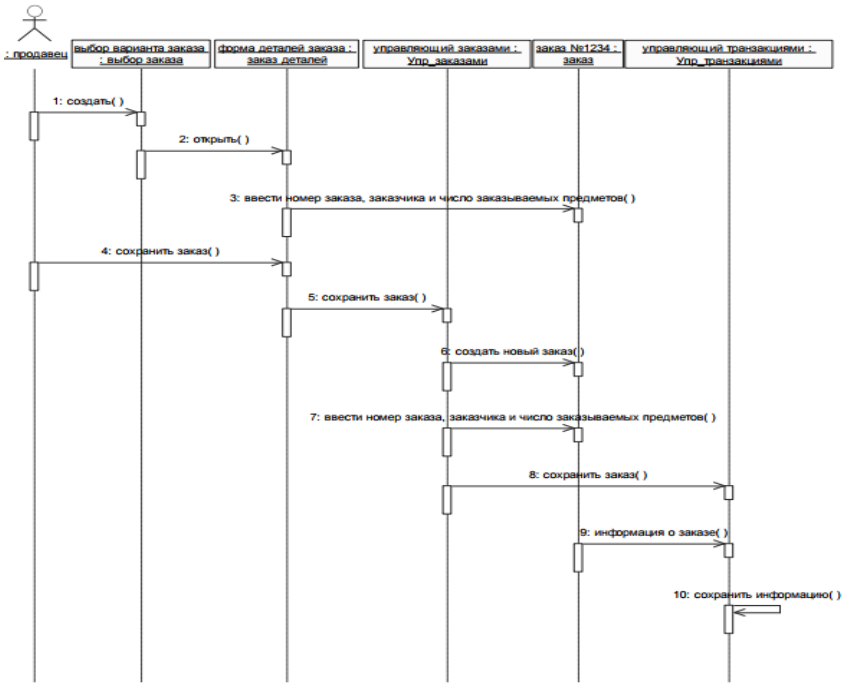


Рисунок 4.3 – Окончательный вид диаграммы последовательности

Вопросы для самоконтроля

1. Перечислите разные типы диаграмм в языке UML.
2. В каких областях науки и техники могут использоваться модели, созданные с помощью программы StarUML?
3. Перечислите основные элементы диаграммы последовательности.
4. Типы сообщений на диаграмме последовательности Какие элементы входят в структуру проекта?
5. Какое расширение имеют файлы проекта в StarUML?
6. Что такое фрейм?
7. Примеры фреймов и их обозначения на диаграмме последовательности.
8. Перечислите основные типы диаграмм в программе StarUML.

§ 3.1 Диаграмма кооперации

Диаграмма кооперации (кооперативная диаграмма) в UML рассматривается как вид диаграммы взаимодействия, ориентированный на визуализацию взаимодействия и отношений между объектами в системе. Главная задача – показать, как объекты обмениваются сообщениями для достижения нужной цели или реализации определенных вариантов использования. В отличие от диаграммы последовательности на данной диаграмме можно показать не только последовательность взаимодействия между объектами, но графически представить все структурные связи между ними.

Диаграммы кооперации отображают поток событий через конкретный сценарий варианта использования, упорядочены по времени, а кооперативные диаграммы больше внимания заостряют на связях между объектами.

Диаграмма кооперации – это альтернативный способ изображения сценария варианта использования. Этот тип диаграмм заостряет внимание на связях между объектами, отображая обмен данными в системе. А диаграммы последовательности отображают взаимодействие объектов во времени, поэтому ее следует читать сверху вниз и слева направо.

Диаграммы кооперации содержат все те же элементы, что и диаграммы последовательности: объекты, действующие лица, связи между ними и сообщения, которыми они обмениваются, но они уже не упорядочены во времени.

Рассмотрим основные элементы, которые необходимы для создания диаграммы кооперации.

К основным элементам относятся объекты, представленные в виде прямоугольников, в которых записаны имя объекта, его класс и, возможно, значения атрибутов.

Связи как линии, соединяющие это объекты, указывают ассоциации. Можно четко определить имена ассоциации и ролей, которые объекты играют в этой ассоциации.

Сообщения как элемент Диаграммы кооперации отправляются от объекта-отправителя к объекту-получателю, и представлены в виде стрелок. Каждая стрелка может иметь метку, указывающую на тип сообщения (например, вызов метода, сигнал).

В отличие от диаграммы последовательности на диаграмме кооперации имеются числовые и ли алфавитные маркеры, которые указывают на порядок потока сообщений – порядковые номера.

Полосы активации представлены тонким прямоугольником на объекте, описывают время активности объекта во время взаимодействия.

На диаграмме кооперации для визуализации сложных взаимодействий могут быть использованы условия и циклы.

Примечания и комментарии могут быть добавлены к любому графическому элементу для большей ясности или для добавления дополнительной информации.

В UML диаграммы кооперации используются для:

Моделирования взаимодействия объектов в конкретном сценарии варианта использования.

Описания альтернативных сценариев в рамках вариантов использования или операций, включающих взаимодействие различных объектов и взаимодействий.

Определения и объяснения ролей объектов, выполняющих определенный поток событий в варианте использования.

Рассмотрим некоторые ограничения при использовании диаграмм кооперации в UML:

Если количество объектов и сообщений увеличивается, диаграмма становится сложнее для чтения.

Трудно предоставить дополнительную описательную информацию, такую как время, точки принятия решений и другие неструктурированные данные.

Один и тот же набор объектов может участвовать в реализации различных кооперации. В зависимости от рассматриваемого взаимодействия, взаимосвязь между отдельными объектами и поток сообщений между ними могут меняться.

§ 3.2 Создание диаграммы кооперации

Постановка задачи. Создать диаграмму кооперации. Конечный вид диаграммы представлен на рисунке 3.1.

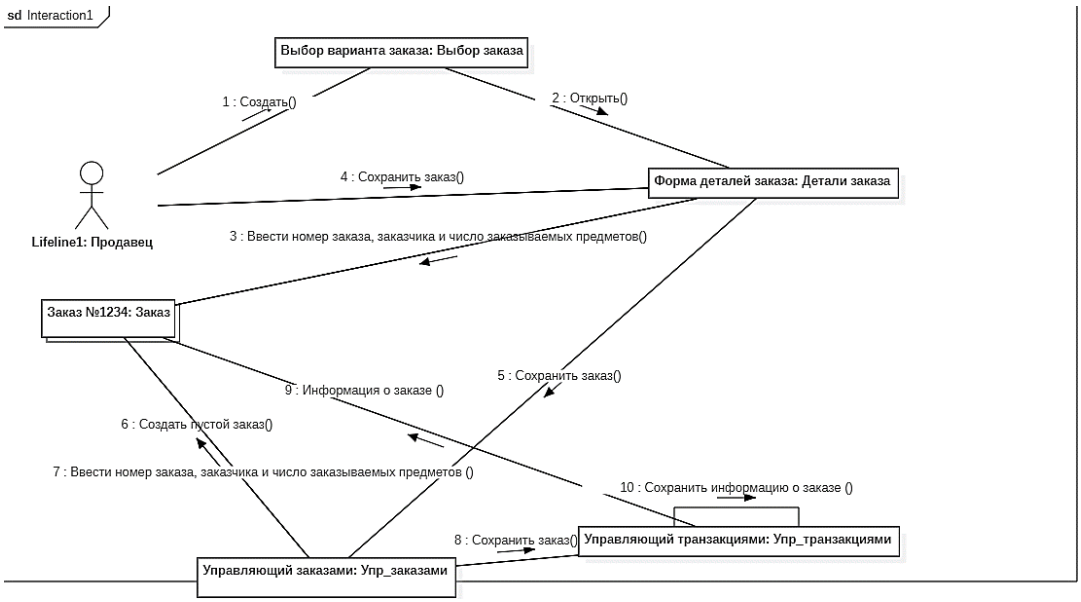


Рисунок 3.1 – Окончательный вид кооперативной диаграммы

Добавление кооперативной (коммуникационной) диаграммы
Открыть или создать модель:
Правой кнопкой мыши (ПКМ) нажать на **Model** или нужном пакете.
Выбрать **Add Diagram → Communication Diagram**(рис. 3.2).

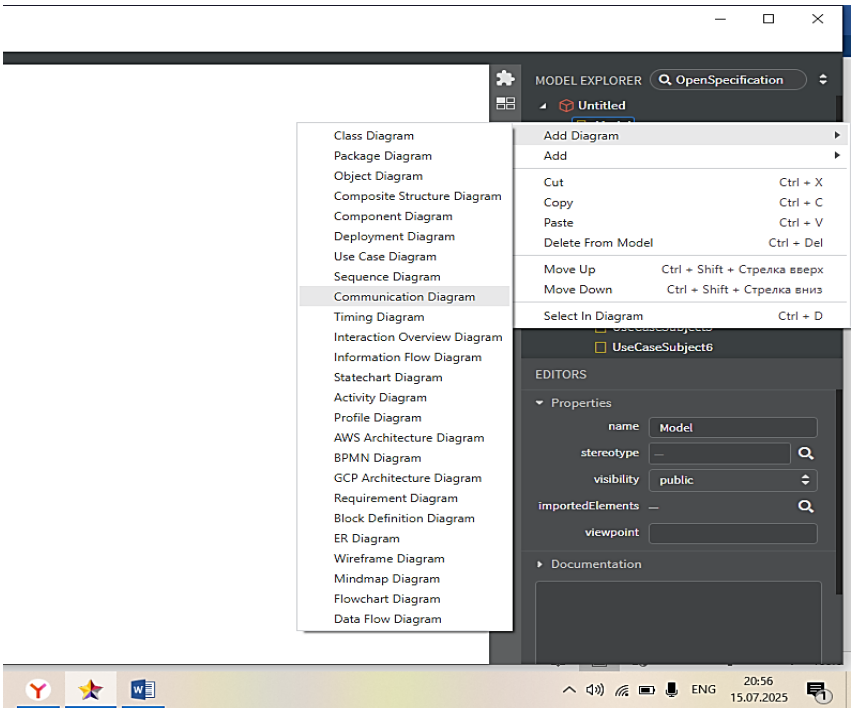


Рисунок 3.2 – Добавление кооперативной диаграммы

Назвать эту диаграмму *Ввод заказа*.
Добавление действующего лица и объектов на диаграмму
1. Перетащить действующее лицо *Продавец* из браузера на диаграмму.
2. Нажать кнопку Lifeline панели инструментов.
3. Щелкнуть мышью где-нибудь внутри диаграммы, чтобы поместить туда новый объект.

4. Назвать объект *Выбор варианта заказа*.

5. Повторив шаги 3 и 4, поместить на диаграмму объекты:

– *Форма деталей заказа*

– *Заказ №1234*

Добавление сообщений на диаграмму

1. На панели инструментов нажать кнопку Connector (Связь объекта).

2. Провести мышью от действующего лица *Продавец* к объекту *Выбор варианта заказа*.

3. Повторить шаги 1 и 2, соединив связями следующие объекты:

- Действующее лицо *Продавец* и объект *Форма деталей Заказа*
- Объект *Форма деталей Заказа* и объект *Выбор Варианта Заказа*
- Объект *Форма деталей Заказа* объект *ЗаказN1234*

4. На панели инструментов нажать кнопку Forward Message (Сообщение связи).

5. Щелкнуть мышью на связи между *Продавец* и *Форма деталей Заказа*.

6. Выделив сообщение, ввести его имя – *Создать новый заказ*;

7. Повторив шаги с 4 по 6, поместить на диаграмму сообщения:

- *Открыть форму* – между *Выбор Варианта Заказа* и *Форма Деталей Заказа*.
- *Ввести номер заказа, заказчика и число заказываемых предметов* – между *Заказ №1234* и *Форма Деталей Заказа*
- *Сохранить заказ* – между *Продавец* и *Форма деталей Заказа*
- *Создать пустой заказ* – между *Форма деталей Заказа* и *Заказ №1234*
- Ввести номер заказа, заказчика и число заказываемых предметов – между

Форма деталей Заказа и *Заказ №1234*

- Сохранить заказ – между *Форма деталей Заказа* и *Заказ №1234*

Теперь нужно поместить на диаграмму дополнительные элементы, а также рассмотреть ответственности объектов.

1. Добавление на диаграмму дополнительных объектов

2. Нажать на кнопку Lifeline панели инструментов.

3. Щелкнуть мышью где-нибудь на диаграмме, чтобы поместить туда новый

4. Объект.

5. Ввести имя объекта – Управляющий заказами.

6. На панели инструментов нажать кнопку Lifeline.

7. Поместить на диаграмму еще один объект.

8. Ввести его имя – Управляющий транзакциями.

Назначение ответственностей объектам

1. Выделить сообщение 5: *Создать пустой заказ*. Выделять слова, а не стрелку.

2. Нажав комбинацию клавиш CTRL+D, удалить это сообщение.
3. Повторить шаги 1 и 2 для удаления сообщений 6 и 7:
- *Ввести номер заказа, заказчика и число заказываемых предметов*
- *Сохранить заказ*
4. Выделить связь между объектами *Форма деталей Заказа* и *Заказ №1234*
5. Нажав комбинацию клавиш CTRL+D, удалить эту связь
6. На панели инструментов нажать кнопку Connector
7. Нарисовать связь между *Форма деталей Заказа* и *Управляющий Заказами*.
8. На панели инструментов нажать кнопку Connector
9. Нарисовать связь между *Управляющий Заказами* и *Заказ №1234*
10. На панели инструментов нажмите кнопку Connector
11. Нарисовать связь между *Заказ №1234* и *Управляющий Транзакцией*.
12. На панели инструментов нажать кнопку Connector
13. Нарисовать связь между *Управляющий Заказами* и *Управляющий Транзакцией*.
14. На панели инструментов нажать кнопку Forward Message
15. Щелкнуть мышью на связи между объектами *Форма деталей Заказа* и *Управляющий Заказами*, чтобы ввести новое сообщение.
16. Назвать это сообщение сохранить заказ.
17. Повторить шаги 14-16, добавив сообщения с шестого по девятое, и назвав

их:

– Создать новый заказ – между *Управляющий Заказами* и *Заказ №1234* *– Ввести номер заказа, заказчика и число заказываемых предметов* – между *Управляющий Заказами* и *Заказ №1234 – Сохранить заказ* – между *Управляющий Заказами* и *Управляющий Транзакцией – Информация о заказе* – между *Управляющий Транзакцией* и *Заказ №1234*

18. На панели инструментов нажать кнопку Reverse Message
19. Щелкнув на объекте *Управляющий Транзакцией*, добавить к нему рефлексивное сообщение.
20. На панели инструментов нажать кнопку Reverse Message
21. Щелкнуть мышью на рефлексивной связи *Управляющий Транзакциями*, чтобы ввести туда сообщение.
22. Назвать новое *Сохранить информацию о заказе в базе данных (рис.3.3)*.

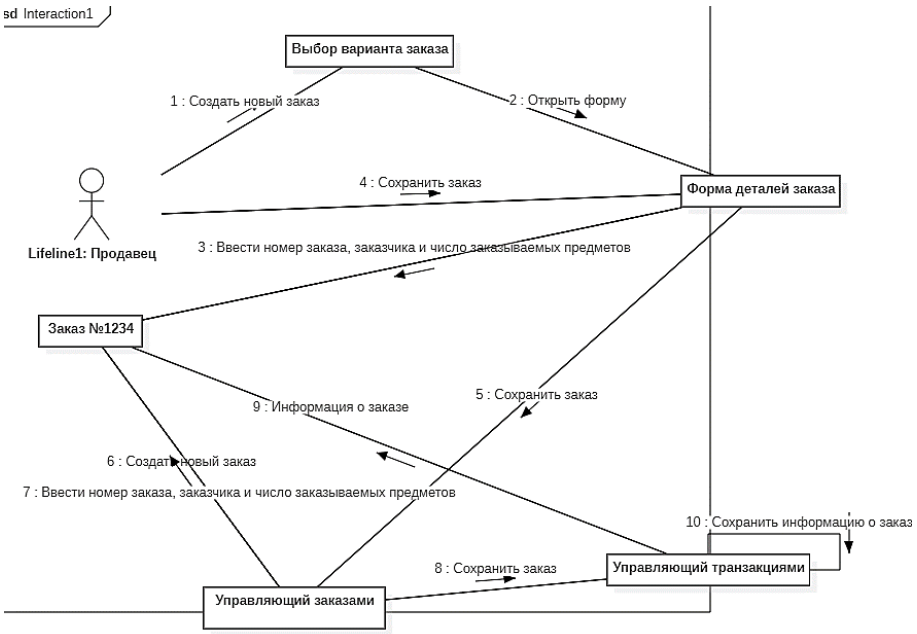


Рисунок 3.3

Соотнесение объектов с классами

1. Щелкнуть левой кнопкой мыши два раза на объекте Форма деталей Заказа.

2. В открывшемся меню выбрать пункт Create type

3. Появится окно спецификации классов.

4. В поле имени ввести *Выбор заказа*.

5. Щелкнуть на кнопке OK, чтобы вернуться к диаграмме. Теперь объект называется *Выбор варианта заказа:Выбор Заказа*

2. Для соотнесения остальных объектов с классами повторить шаги с 1 по 7:

– Класс *Детали заказа* соотнесите с объектом *Форма деталей заказа*

– Класс *Упр_заказами* – с объектом *Управляющий заказами*

– Класс *Заказ* – с объектом *ЗаказN1234*

– Класс *Упр_транзакциями* – с объектом *Управляющий транзакциями*

Соотнесение сообщений с операциями

1. Щелкнуть мышкой два раза на сообщении 1: *Создать новый заказ()*.

1. В открывшемся меню выбрать пункт CreateOperation. Появится окно спецификации операции.

2. В поле имени введите имя операции – *Создать()*.

3. Нажать на кнопку OK, чтобы закрыть окно спецификации операции и вернуться к диаграмме.

4. Еще раз щелкните правой кнопкой мыши на сообщении 1.

5. В открывшемся меню выбрать пункт OpenSpecification (Открыть спецификацию).

6. В раскрывающемся списке Name (Имя) укажите имя новой операции.

7. Нажать на кнопку OK.

8. Повторить шаги 1-8, чтобы создать новые операции и соотнести с ними остальные сообщения:

– Сообщение 2: *Открыть форму* соотнесите с операцией *Открыть()*

– Сообщение 3: *Ввести номер заказа, заказчика и число заказываемых предметов* - с операцией *Ввести номер заказа, заказчика и число заказываемых предметов()*

– Сообщение 4: *Сохранить заказ* – с операцией *Сохранить заказ()*

– Сообщение 5: *Сохранить заказ* – с операцией *Сохранить заказ()*

– Сообщение 6: *Создать пустой заказ* – с операцией *Создать пустой заказ()*

– Сообщение 7: *Ввести номер заказа, заказчика и число заказываемых предметов*- одноименной операцией

– Сообщение 8 Сохранить заказ – с операцией *Сохранить заказ()*

– Сообщение 9 *Информация о заказе* – с одноименной операцией

– Сообщение 10 *Сохранить информацию о заказе* с одноименной операцией Ваша диаграмма должна выглядеть, как показано на рисунке 2.

Вопросы для самоконтроля

1. Какие элементы может содержать диаграмма кооперaций?
2. На каком этапе разработки программной системы разрабатывается диаграммы кооперации? Каково ее назначение?
3. Перечислите ограничения при использовании диаграмм кооперации в UML

§ 4. 1 Диаграммы состояний

Для представления взаимодействий объектов в системе используют диаграммы последовательности и кооперации. Прецеденты и сценарии описывают поведение системы, но если нам важна динамика класса, изменение состояний отдельного объекта, то для этой цели полезно построить диаграммы состояний.

Диаграмма состояний показывает положение одиночного объекта, события и сообщения, которые вызывают переход из одного состояния в другое, и действия, являющиеся результатом смены состояния. Диаграмма состояний показывает объект с момента его создания и до его уничтожения.

Диаграмму состояний строят не для каждого класса в системе, а только для классов с динамическим поведением, которые отсылают и принимают много сообщений, изменяют свое состояние. Программный код из диаграмм состояний не генерируются, но они важны для понимания динамики поведения класса, дают возможность понять логику изменений перед кодированием.

Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

Процессы, происходящие, когда объект находится в определенном состоянии, называются действиями (actions).

Рассмотрим основные элементы диаграмм состояний**.**

Диаграмму состояний часто рассматривают в контексте конечного автомата. Тогда можно сказать, что ***диаграмма состояний (State chart diagram)*** показывает автомат, фокусируя внимание на потоке управления от состояния к состоянию.

Автомат (Statemachine) – это описание последовательности состояний, через которые проходит объект на протяжении всего жизненного цикла, реагируя на события, – в том числе описание реакций на эти события.

Состояние (State) –это ситуация в жизни объекта, на протяжении которой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого-то события.

Для поиска состояний класса можно просматривать атрибуты этого класса. Хорошим индикатором состояний является такой атрибут класса как «статус».

Диаграмма состояний изображается в виде графа с вершинами и ребрами.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами. Под именем состояния могут размещаться действия (рис. 4.1).

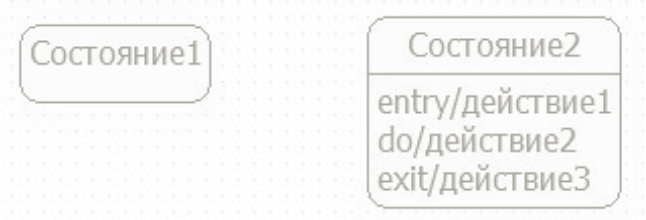


Рисунок 4.1 – Состояния

Кроме обычных состояний на диаграмме состояний могут размещаться псевдосостояния.

Псевдосостояние (pseudo-state) **–** вершина в конечном автомате, которая имеет форму состояния, но не обладает поведением.

Примерами псевдосостояний, которые определены в языке UML, являются начальное и конечное состояния.

Начальное состояние (startstate) **–** разновидность псевдосостояния, обозначающее начало выполнения процесса изменения состояний конечного автомата или нахождения моделируемого объекта в составном состоянии.

В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка, из которого может только выходить стрелка-переход (рисунок 4.2)

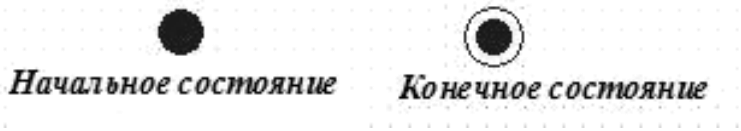


Рисунок 4. 2 – Начальное и конечное состояния

Конечное состояние (finalstate) **–** разновидность псевдосостояния, обозначающее прекращение процесса изменения состояний конечного автомата (рис. 4.2).

Объект находится в конечном состоянии непосредственно перед уничтожением. На диаграмме состояний может быть одно и только одно начальное состояние. В то же время, может быть столько конечных состояний, сколько нужно, или их может не быть вообще.

Находясь в каком-либо состоянии, объект может выполнять определенные действия, с состоянием можно связать такие действия как входное и выходное действия и внутренняя или просто деятельность.

Деятельность

Деятельностью (activity) называется поведение, реализуемое объектом, пока он находится в данном состоянии. Деятельность – это прерываемое поведение. Оно может выполняться до своего завершения, пока объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

Входное действие

Входным действием (entry action) называется поведение, которое выполняется, когда объект переходит в данное состояние. Данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть этого перехода. В отличие от деятельности, входное действие рассматривается как непрерываемое. Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

Выходное действие

Выходное действие (exit action) подобно входному. Однако, оно осуществляется как составная часть процесса выхода из данного состояния. Оно является частью процесса такого перехода. Как и входное, выходное действие является непрерываемым.

Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Поведение объекта во время деятельности, при входных и выходных действиях может включать отправку события другому объекту. В этом случае описанию деятельности, входного действия или выходного действия предшествует знак « ^ ».

Соответствующая строка на диаграмме выглядит как

Do: ^Цель.Событие (Аргументы)

Здесь Цель – это объект, получающий событие, Событие – это посылаемое сообщение, а Аргументы являются параметрами посылаемого сообщения.

В общем случае действие имеет следующий формат:

<метка действия / выражение действия>

Деятельность может также выполняться в результате получения объектом некоторого события. При получении некоторого события выполняется определенная деятельность.

Срабатывание <перехода> (fire) – выполнение перехода из одного состояния в другое.

На диаграмме **переход изображается сплошной стрелкой.** У перехода существует несколько спецификаций: событие, сторожевое условия, действия и посылаемые события.

<имя события>(<список параметров, разделенных запятыми>) [<сторожевое условие>]/<выражение действия>.

Событие (event) – это то, что вызывает переход из одного состояния в другое. Событие размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции, так и обычную фразу.

Большинство переходов должны иметь события, так как именно они, прежде всего, заставляют переход осуществиться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

Сторожевое условия

Сторожевое условие (guard) определяют, когда переход может, а когда не может осуществиться. В противном случае переход не осуществится.

Сторожевое условие изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки.

Сторожевое условие задавать необязательно. Однако если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет автоматически выбран.

Выражение действия (action expression) представляет собой вызов операции или передачу сообщения, имеет атомарный характер и выполняется сразу после срабатывания соответствующего перехода до начала действий в целевом состоянии.

Выражение действия выполняется в том и только в том случае, когда переход срабатывает. Атомарность действия означает, что оно не может быть прервано никаким другим действием до тех пор, пока не закончится его выполнение. Данное действие может оказывать влияние как на сам объект, так и на его окружение, если это с очевидностью следует из контекста модели. Данное выражение записывается после знака "/" в строке текста, присоединенной к соответствующему переходу.

В общем случае, выражение действия может содержать целый список отдельных действий, разделенных символом ";". Обязательное требование - все действия из списка должны четко различаться между собой и следовать в

порядке их записи. На синтаксис записи выражений действия не накладывается никаких ограничений. Главное – их запись должна быть понятна разработчикам модели и программистам. Поэтому чаще всего выражения записывают на одном из языков программирования, который предполагается использовать для реализации модели.

§ 4.2 Создание диаграммы состояний

Постановка задачи

Построить диаграмму Состояний для класса *Заказ*, показанную на рисунке 4.3

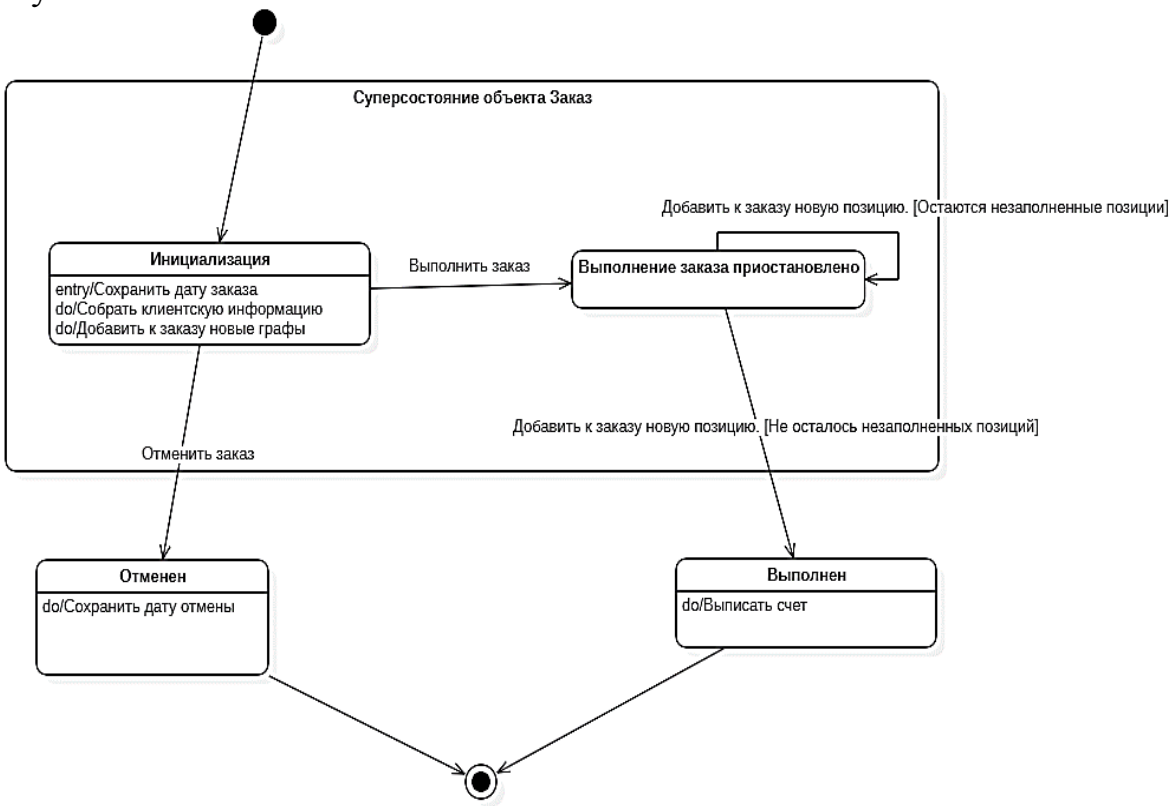


Рисунок 4.3 – Диаграмма состояний для класса *Заказ*

1. Найти в браузере класс *Заказ*.

2. Щелкнуть на классе правой кнопкой мыши и в открывшемся меню указать пункт AddDiagram>StatechartDiagram (Создать диаграмму состояний) (рисунок 4.4)

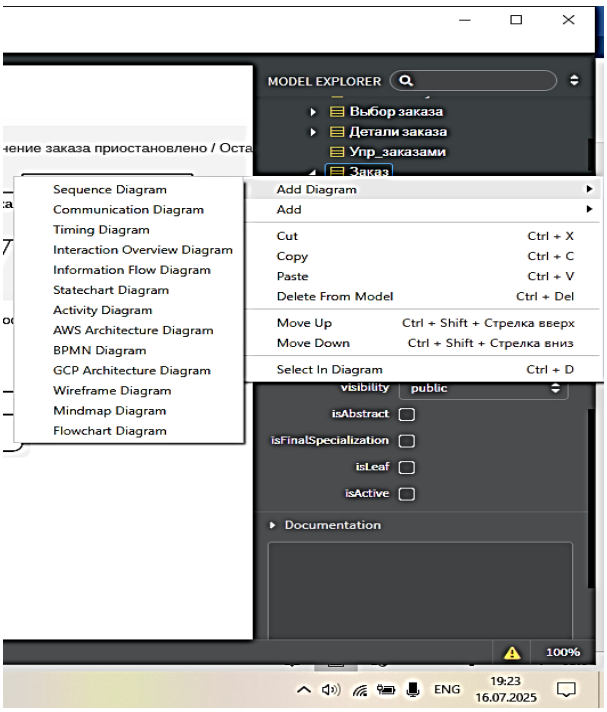


Рисунок 4.4 – Создание диаграммы состояний

Добавление начального и конечного состояний

1. Нажать кнопку InitialState (Начальное состояние) панели инструментов.
2. Поместить это состояние на диаграмму.
3. Нажать кнопку FinalState (Конечное состояние) панели инструментов.
4. Поместить это состояние на диаграмму.

Добавление суперсостояния

1. Нажать кнопку Simple State (Состояние) панели инструментов.
2. Поместить это состояние на диаграмму.

Добавление оставшихся состояний

1. На панели инструментов нажать кнопку Simple State (Состояние).
2. Поместить состояние на диаграмму.
3. Назвать состояние *Отменен*.
4. На панели инструментов нажать кнопку Simple State (Состояние).
5. Поместить состояние на диаграмму.
6. Назвать состояние *Выполнен*.
7. На панели инструментов нажать кнопку Simple State (Состояние).
8. Поместить состояние на диаграмму внутрь суперсостояния.
9. Назвать состояние *Инициализация*. (рис.6)
10. На панели инструментов нажать кнопку Simple State (Состояние).
11. Поместить состояние на диаграмму внутрь суперсостояния.
12. Назвать состояние *Выполнение заказа приостановлено*.

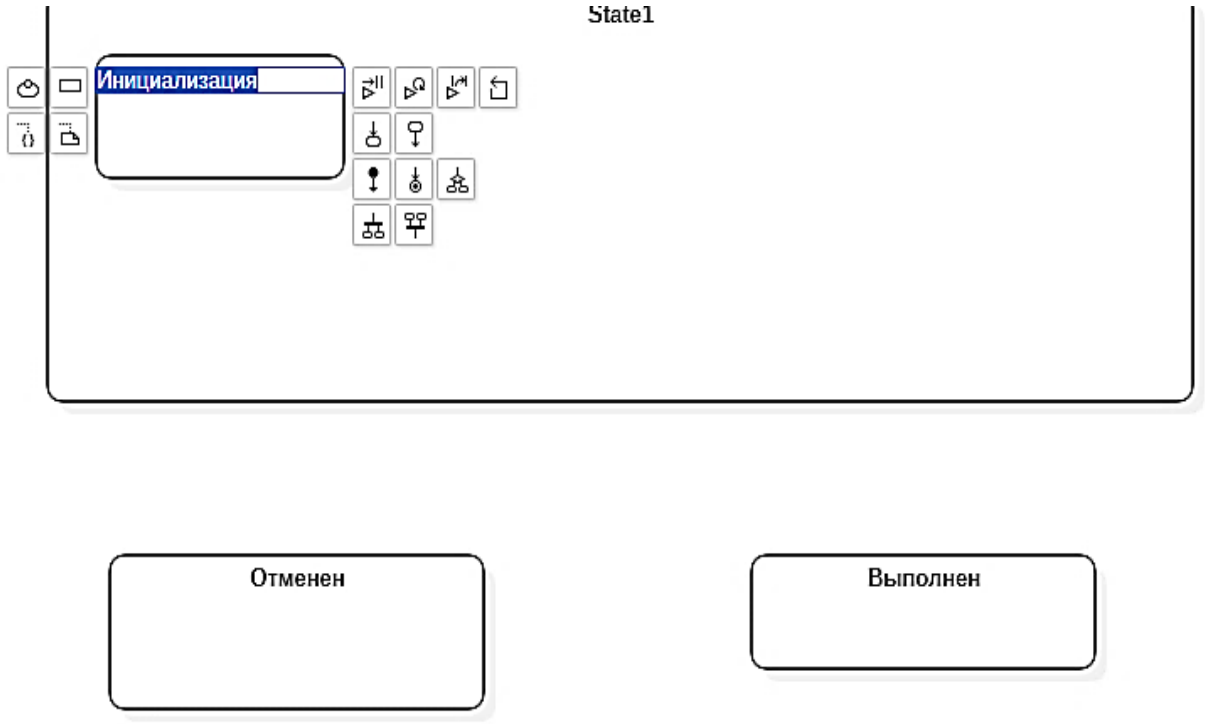


Рисунок 4.5 – Добавление списка внутренних действий

Описание состояний

1.Дважды щелкнуть на состоянии Инициализация
2.Выбрать тип нового действия Add Entry Activity
3.Выбрать Activity
4.Назвать его Сохранить дату заказа (рис.4.6-4.7)

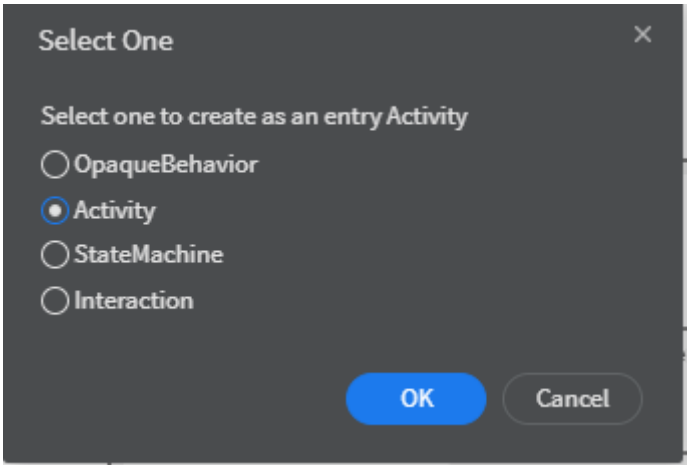


Рисунок 4.6 – Добавление действия Add Entry Activity

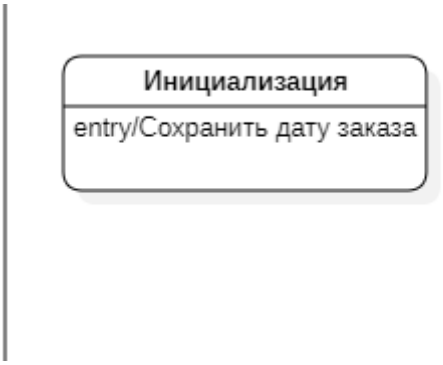


Рисунок 4.7 – Добавление действия Entry Activity

5. Повторить этапы 1-4, добавив следующие действия:

6.Собрать клиентскую информацию, Add Do Activity

7.Добавить к заказу новые графы, Add Do Activity

8. Дважды щелкнуть на состоянии Отменен.

Повторить этапы 1-4, добавив действие Сохранить дату отмены, Add Entry Activity

9. Дважды щелкнуть на состоянии Выполнен.

10. Повторить этапы 1-4, добавив действие Выписать счет, указать пункт Add Do Activity.

Добавление переходов

1. Нажать кнопку Transition (Переход) панели инструментов.
2. Щелкнуть мышью на начальном состоянии.

3. Провести линию перехода к состоянию *Инициализация*.

4. Повторив шаги с первого по третий, создать следующие переходы:

- От состояния Инициализация к состоянию *Выполнение заказа приостановлено*
- От состояния *Выполнение заказа приостановлено* к состоянию *Выполнен*
- От **суперсостояния** к состоянию *Отменен*
- От состояния *Отменен* к **конечному** состоянию
- От состояния *Выполнен* к **конечному** состоянию

5. На панели инструментов нажать кнопку SelfTransition (Переход к себе).

6. Щелкнуть мышью на *состоянии Выполнение заказа приостановлено*

Описание переходов

1. Дважды щелкнув мышью на переходе от состояния *Инициализация* к состоянию *Выполнение заказа приостановлено*, открыть окно спецификации перехода (рисунок 4.8).

2. Ввести фразу *Выполнить заказ*.

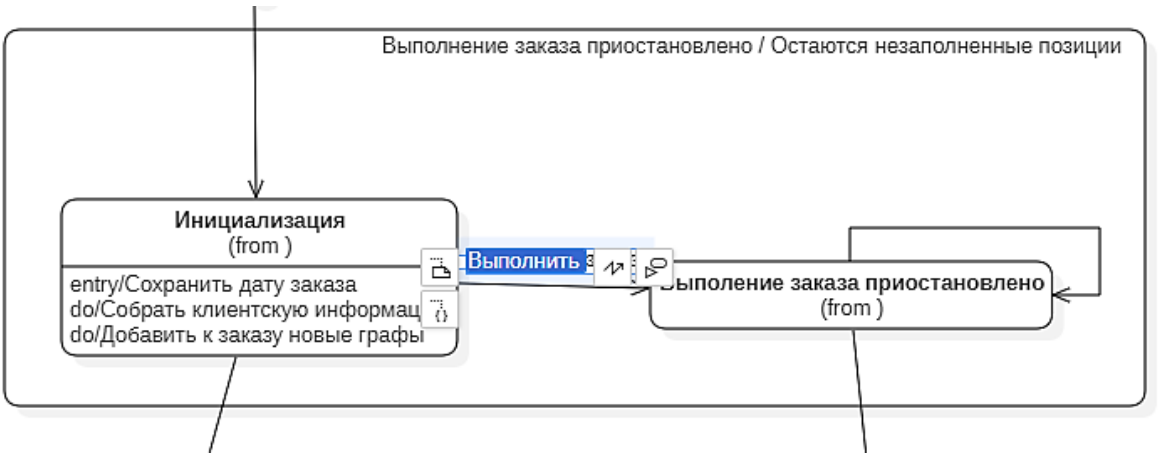


Рисунок 4.8 – Окно спецификации перехода

3. Щелкнув на кнопке ОК, закрыть окно спецификации.

4. Повторив шаги с первого по третий, добавить событие *Отменить заказ* к переходу между суперсостоянием и состоянием *Отменен*.

5. Дважды щелкнув мышью на переходе от состояния *Выполнение заказа приостановлено* к состоянию *Выполнен*, открыть окно его спецификации.

6. Ввести фразу *Добавить к заказу новую позицию*.

7. Перейти на вкладку Properties (Свойства) (рис.4.9)

8. В поле Guard (Сторожевое Условие) ввести *Не осталось незаполненных позиций*.

9. Щелкнув на кнопке ОК, закрыть окно спецификации.

10. Дважды щелкнуть мышью на рефлексивном переходе (SelfTransition) состояния *Выполнение заказа приостановлено*.

11. Ввести фразу *Добавить к заказу новую позицию*.

12. Перейти на вкладку Properties (Свойства).

13. В поле Guard (Сторожевое Условие) ввести *Остаются незаполненные позиции*.

14. Щелкнуть на кнопке ОК, закрыть окно спецификации.

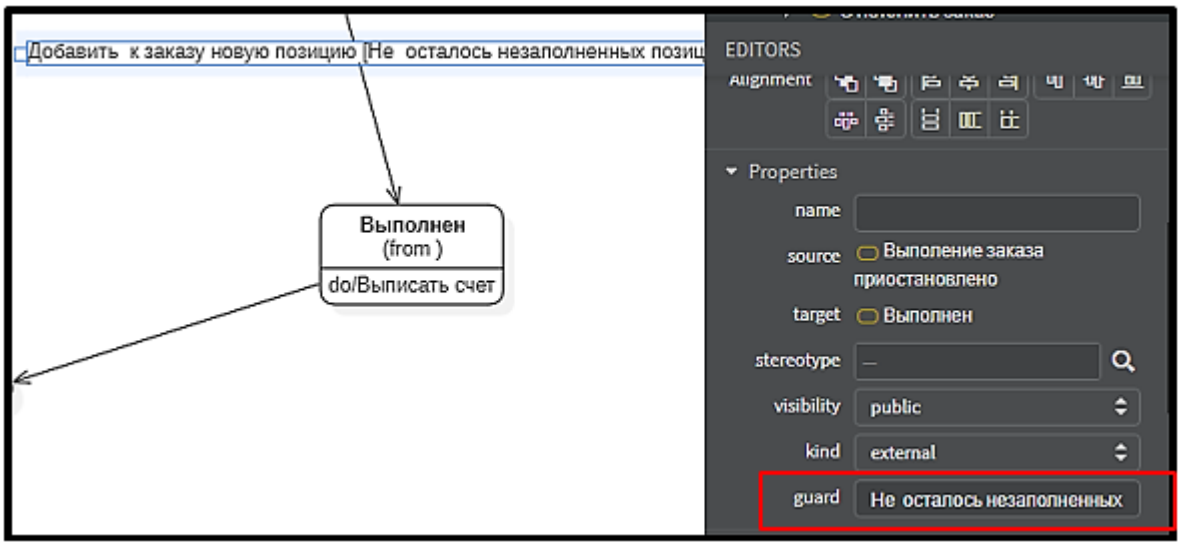


Рисунок 4.9 – Окно Properties

Вопросы для самоконтроля

1Для чего предназначены диаграммы состояний.
2Основные элементы диаграммы состояний, их характеристики

§ 5.1 Диаграмма активности (Activity diagram)

На уроках информатики рисуют блок-схемы, чтобы наглядно изобразить алгоритм решения некоторой задачи. Действительно, моделируя поведение проектируемой системы, часто недостаточно изобразить процесс смены ее состояний, а нужно также раскрыть детали алгоритмической реализации операций, выполняемых системой. Для этой цели традиционно использовались блок-схемы или структурные схемы алгоритмов. В UML для этого существуют диаграммы деятельности или активности, являющиеся частным случаем диаграмм состояний. Диаграммы деятельности удобно применять для визуализации алгоритмов, по которым работают операции классов.

Алгоритм – последовательность определенных действий или элементарных операций, выполнение которых приводит к получению желаемого результата.

Диаграмма деятельности – это графическое представление рабочих процессов поэтапных действий и действий с поддержкой выбора, итерации и параллелизма. В сущности, это блок-схем, показывающая, как поток управления переходит от одной деятельности к другой, при этом внимание фиксируется на результате деятельности

Обозначения на диаграмме активности также напоминают те, которые встречаются на блок-схеме, хотя есть, и некоторые существенные отличия. С другой стороны, нотация диаграмм активности очень похожа на ту, которая используется в диаграммах состояний.



Рисунок 5.1 – Диаграмма деятельности оформления заказа в интернет-магазине

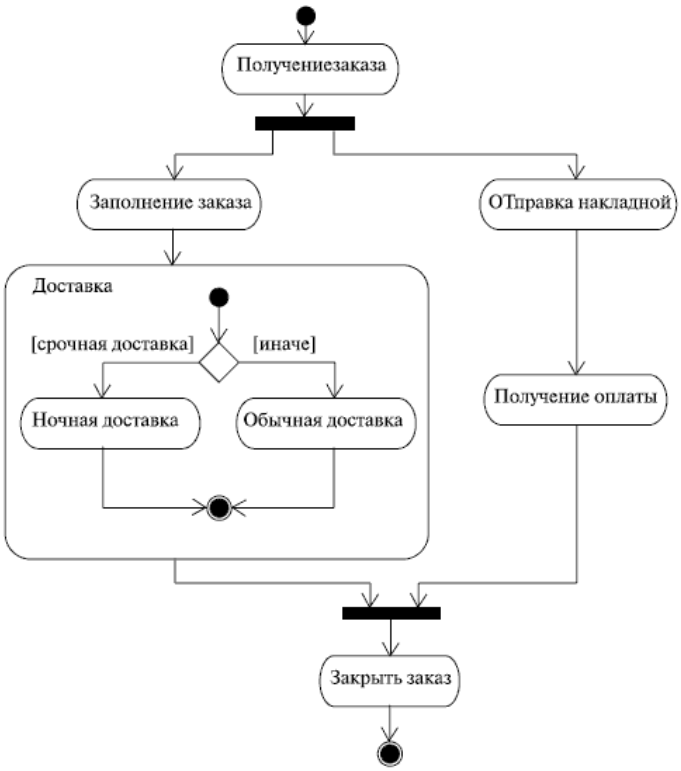


Рисунок 5.2 – Диаграмма деятельности доставки заказа

Главное отличие между Activity и Simple State chart в том, что в первом случае основное – действия, а во втором – статичное состояние. При этом Activity diagram больше подходит для моделирования последовательности действий, а Simple State chart для моделирования дискретных состояний объекта.

Графически диаграмма деятельности представляется в виде графа, вершинами которого являются состояния действия, дугами – переходы от одного состояния действия к другому

Рассмотрим некоторые элементы диаграммы активности.

Action (активность) обозначает выполняемые задачи или выполнение определенных действий в течение жизни объекта и обозначается прямоугольником со скругленными углами.

Начальное состояние, как и в диаграмме состояний указывает на начало выполнения процесса и обозначается черным кругом.

Поток управления/переход (Control Flow) – указывает переход из одного состояния в другое или по завершении выполнения определенного действия в начало другого и обозначается стрелкой.

Параллельное выполнение (Fork Node) позволяет определить независимо выполняемые действия. При этом действия разделяются на несколько выполняемых независимо, и только по завершении всех действий объект продолжает работу. Этот значок представляет собой горизонтальную или вертикальную черту, из которой выходят несколько стрелок.

Решение/Ветвление (Decision Node) позволяет показать зависимость дальнейшей работы от внешних условий или решений. Этот значок

аналогичен командам языка программирования IF и else может иметь больше двух выходов, но обычно используют выбор из двух переходов, т.е. это точка выбора между несколькими вариантами развития процесса, обозначается ромбом.

Объединение (Merge Node)- это слияние нескольких веток в одну, позволяет вернуться к общему сценарию после выполнения альтернативных путей.

Синхронизация (Join Node) – слияние нескольких параллельных процессов в один поток, обозначается линией, в которую входят несколько стрелок.

Swimlanes (дорожки) – это визуальный элемент на диаграмме активности, который направлен на группировку действий по ролям, ресурсам, элементам организации или местоположению. Изображается с помощью прямоугольников. Название дорожки отделяется от ее содержания линией. Использование дорожки визуально отделяет каждую группу от соседней вертикальной черты (как плавательные дорожки в бассейне). Каждой дорожке присваивается свое имя, и каждая дорожка представляет сферу ответственности за часть всей работы, изображенной на диаграмме.

Далее мы рассмотрим основные правила построения диаграммы деятельности.

Диаграмма должна начинаться с начального состояния, в котором начинаются поток или потоки при вызове деятельности извне, и завершаться конечным состоянием, который останавливает все потоки данной деятельности.

Для реализации параллельных потоков необходимо использовать точки разделения и точки слияния:

– из точки разделения (Fork Node) выходят два и более потоков, выполняющихся параллельно;

– точка слияния (Join Node) синхронизирует потоки: каждый из них ждет этой точки остальными потоками, после чего продолжается последовательное исполнение.

§ 5.2 Построение диаграммы активности для варианта использования «Выполнить поставку Заказа»

Описание варианта использования.

При оформлении заказа проверяют каждую содержащуюся в нем позицию, чтобы убедиться в наличии соответствующих товаров на складе. После этого выписываются товары для реализации заказа. Во время выполнения этих процедур одновременно проверяется прохождение платежа. Если платеж прошел, и товары имеются на складе, то осуществляется их поставка. Если платеж прошел, но товары на складе отсутствуют, то заказ ставится в ожидание. Если платеж не прошел, то заказ аннулируется.

1. Найти в браузере вариант использования «Выполнить поставку заказа»

2. Щелкнуть на классе правой кнопкой мыши и в открывшемся меню указать пункт AddDiagram>ActivityDiagram (Создать диаграмму активности).

3. Назвать диаграмму «Выполнить поставку» и откройте ее двукратным щелчком мыши

4. На панели инструментов щелкнуть мышкой на элементе Swimlane (Horizontal), затем на поле диаграммы.

5. Открыть окно Properties, изменить поле Name на *Клерк.*

6. Выполнить заново пункты 4-5 и присвоить полю Name *Бухгалтерская система.*

7. Найти в браузере сплошной черный кружок (начальное состояние). Перенести его на дорожку *Клерк.*

8. Выбрать из панели инструментов объект Action и поместить его на диаграмму в "дорожку" *Клерк*. Изменить имя объекта на "*Получить заказ*".

9. Повторить предыдущий этап, создать на «дорожке» *Клерк* 4 новых Action и присвоить им имена *Проверить позицию заказа, закрепить позицию за заказом, Поставить заказ в ожидание, Скомплектовать заказ*

10. Поместить на «дорожку» 2 новых объекта Final (конечное состояние). Одному из них изменить поле Name на «*Выполнить поставку*»

11. На дорожку *Система* поместить новый объект Activity и присвоить полю Name "*Проверить платеж*". На эту же дорожку поместите новый объект Final и изменить его свойство Name на «*Отменить заказ*».

12. Поместить на «дорожку» Клерк 2 объекта Fork. Присвоить полю Name спецификации одного объекта «*1*», другого- «*2*».

13. Поместить на «дорожку» *Клерк* объект Dicision (выбор), Присвоить Свойству Name «*Позиция имеется?*».

Поместить на «дорожку» *Система* объект Dicision. Присвоить свойству Name «*Деньги поступили?*»

14. Щелкнуть мышкой на панели инструментов объекте- стрелке Control Flow (состояние перехода). Затем щелкнуть мышкой на диаграмме объекта начальное состояние. Удерживая кнопку мыши перенести курсор на активность "*Получить заказ*"и лишь затем отпустить курсор. В результате два объекта будут соединены стрелкой.

15. Выполнить этап 14, соединив стрелкой объект Активность «*Получить заказ*» с объектом Fork 1.

16. Соединить этими же стрелками объекты 1 и «Проверить платеж», 1 и «Проверить позицию заказа», «Проверить заказ» и «Деньги подступили?», «Деньги поступили?» и «Отменить заказ», «Проверить позицию заказа» и «Позиция имеется», «Позиция имеется» и «Закрепить позицию за заказом», «Деньги получены?» и 2, «Закрепить позицию за заказом» и 2, «Позиция имеется?» и «Поставить заказ в ожидание», 2 и «Скомплектовать заказ», «Скомплектовать заказ» и «Выполнить поставку», «Поставить заказ в ожидание» и объект Конечное состояние (без имени)

17. Присвоить некоторым стрелкам наименование (условие перехода).

Для этого, установить курсор на стрелке, соединяющей «Деньги получены?» и «Отменить заказ». Двукратным щелчком мыши открыть окно инициализации, введите «Нет».

18. Выполнить пункт 17 для стрелки, соединяющей «Деньги получены?» и 2 и присвоить значение «Да». Аналогично для стрелки, соединяющей «Позиция имеется?» и «Закрепить позицию за заказом» присвоить «Да». Стрелке, соединяющей «Позиция имеется?» и «Поставить заказ в ожидание» – «Нет».

19. Путем перемещения объектов (установить курсор мыши- нажать-тащить-отпустить) привести диаграмму к виду, показанному на рисунке 5.3.

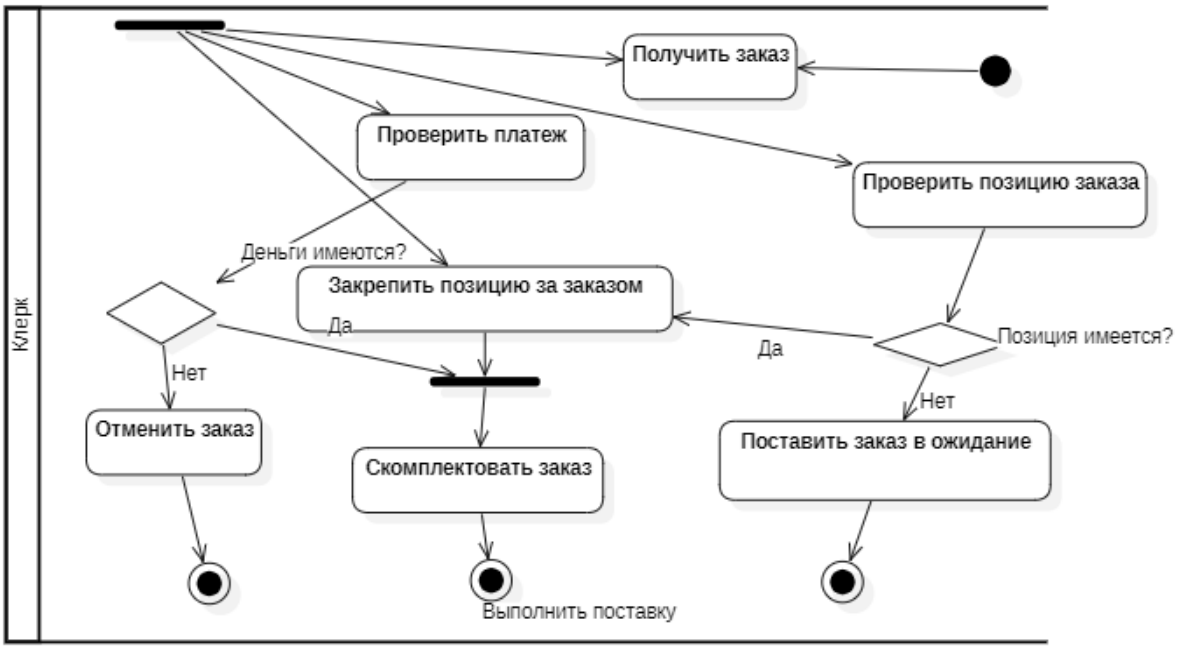


Рисунок 5.3 – Диаграмма активности для варианта использования «Выполнить поставку заказа»

Вопросы для самоконтроля

1. Для чего используется диаграмма Activity?
2. Для чего больше подходит Activity diagram?
3. Какой элемент обозначает выполняемые задачи или выполнение определенных действий в течение жизни объекта?
4. Какой элемент позволяет показать зависимость дальнейшей работы от внешних условий или решений?
5. Каково основное назначение (цели создания) диаграмм действий?
6. Перечислите элементы диаграммы активности.
7. Главное отличие между Activity diagram и Simple State chart.

§ 6.1 Создание диаграмм классов

Диаграмма классов является частью логической модели системы и представляет статическую картину системы.

Как мы отмечали выше, класс – это описание набора объектов с общими атрибутами, операциями и семантикой. Графически класс представляется в виде прямоугольника, разделённого на три части, где записываются его имя, атрибуты и операции.

Для каждой системы строится не одна, а несколько диаграмм классов: возможно, что для каждого прецедента или сценария своя. На одних показывают подмножества классов, объединенные в пакеты, и отношения между ними, на других – отображают те же подмножества, но с атрибутами и операциями классов. Для представления системы разрабатывается столько диаграмм классов, сколько потребуется.

Основные элементы диаграмм классов: объект, состояние, поведение, индивидуальность.

Дадим некоторые определения и опишем основные элементы нотации диаграмм классов.

Объект – это некоторая сущность реального мира или концептуальная (абстрактная) сущность. В качестве объектов могут рассматриваться сотрудник организации, номер дома по улице Ленина, банковский счет клиента и др. Объект имеет четко определенные границы и значение для системы и характеризуется состоянием, поведением и индивидуальностью.

Состояние объекта–это одно из условий, в котором он может находиться. Состояние обычно изменяется со временем и характеризуется набором свойств, которые называются атрибутами. Клиент банка определяется именем, адресом, телефоном, датой рождения т.д.

Поведение определяет, как объект реагирует на запросы других объектов и что может делать сам объект. Поведение характеризуется операциями объекта. Клиент может внести деньги, снять деньги, перевести, оплатить товары и т.д.

Индивидуальность означает, что каждый объект уникален, даже если его состояние идентично состоянию другого объекта. Объекты Иванов Иван и Петров Александр уникальны, хотя каждый из них является покупателем магазина и имеет одинаковые поведение и состояния.

Обычно в системе много объектов, обладающих одинаковым поведением и находящихся в одинаковых состояниях. Например, сотрудники банка, которых может быть несколько десятков, и данные о которых хранятся в базе данных, имеют одинаковые характеристики – фамилия, имя, отчество, дата рождения, должность и т. д. – с разными значениями этих атрибутов, а также могут демонстрировать схожее поведение – подавать заявление на отпуск или перевод в другой отдел. Для группировки объектов используются классы, описывающие группу объектов, обладающих общими свойствами (атрибутами), поведением (операциями), отношениями с другими объектами и семантикой.

Каждый класс – это шаблон для создания объекта. И каждый объект

является экземпляром класса. Важно помнить, что каждый объект может быть экземпляром только одного класса. В контексте банка можно сгруппировать сотрудников, описав общий класс сотрудников. Объект этого класса, Иванов Иван, может содержать следующие данные: полное имя, адрес, должность, зарплата, а также этот объект может уйти в отпуск.

В нотации UML классы и объекты описываются в виде прямоугольников. Прямоугольник класса всегда делится на три части (секции): в первой части указывается имя класса, каждое слово в имени класса обычно пишется с заглавной буквы. Вторая и третья секции могут содержать атрибуты и операции класса соответственно, и эти секции могут быть пустыми. Имена классов выбираются в соответствии с концепциями предметной области. Должно быть одно существительное или словосочетание, наиболее точно характеризующее предмет. Класс должен описывать только одну сущность



Рисунок 6.1 – Изображение классов и объектов

Имя класса может быть простым, как это показано на рисунке 1, или составным (рис. 6.2). Составное имя класса состоит из самого имени класса и из имени пакета, которому принадлежит класс, разделенных двоеточием. Имя класса должно быть уникальным внутри пакета.

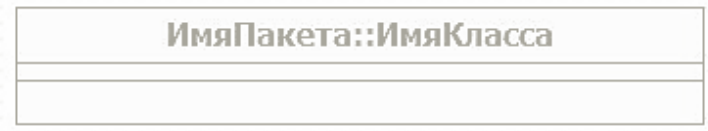


Рисунок 6.2 – Составное имя класса

Составное имя объекта также состоит из имени объекта и имени класса, разделенных двоеточием. Объект может быть анонимным, если неизвестно его настоящее имя. Тогда на диаграмме объект изображается с именем, которое состоит из двоеточия и имени класса, которому принадлежит объект. Если пока неизвестен класс, экземпляром которого является объект, то изображается имя объекта после которого идет двоеточие. Такой объект называется «сиротой» (рис.6. 3).

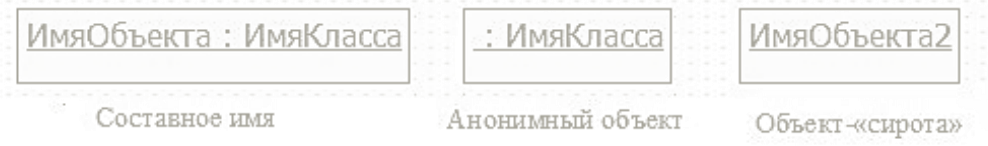


Рисунок 6.3 – Именование объектов

Пример. Класс Сотрудники объект этого класса-некоторого сотрудника – можно изобразить так, как показано на рисунке 6.4.



Рисунок 6.4 – Класс и его объект

Объекту класса Сотрудник дано имя, совпадающее с именем класса.

Рассмотрим некоторые правила выявления классов.

Выявление классов можно начать с изучения потока событий. Имена существительные в описании этого потока дадут понять, что может являться классом. В общем случае существительное может оказаться действующим лицом, классом, атрибутом класса или выражением, не являющимся ни действующим лицом, ни классом, ни атрибутом класса.

Если в ходе проектирования системы уже построили диаграммы взаимодействия, перед тем, как приступать к построению диаграмм классов, необходимо найти на этих диаграммах похожие объекты. Например, на диаграмме последовательности, описывающей оформление заказа объектами Ивановым и Петровым надо обратить внимание на эти объекты: они имеют одинаковые свойства: имя, счет в банке и т.п. Значит, в системе должен появиться класс с именем Покупатель, который будет шаблоном объектов Иванов и Петров.

Некоторые возможные классы будут выявлены при рассмотрении трех стереотипов: сущность (entity), граница (boundary) и управление (control). В рамках изучения темы отношения на диаграммах вариантов использования мы рассматривали стереотипы отношений. Тот же принцип создания нового типа на основе уже существующего применим и для классов.

Стереотип – это механизм, позволяющий категоризировать классы. Он используется для создания нового типа элемента, в данном случае нового типа класса. Например, вы хотите выделить все экранные формы в модели. Для этого нужно создать стереотип Form (Форма).

Стереотипы помогают лучше понять ответственности каждого класса в модели, категоризировать выполняемые ими функции. В UML для этого применяют три основных стандартных вида стереотипов классов: классы-сущности, граничные классы и управляющие классы.

Класс-сущность содержит информацию, хранимую постоянно. Используется для моделирования данных и поведения с длинным жизненным циклом. Они могут представлять информацию о предметной области, а могут представлять элементы самой системы. Часто являясь абстракциями предметной области, они имеют наибольшее значение для пользователя, поэтому в их названиях применяются термины предметной области. Если существует проект базы данных, то можно обратиться к изучению названий таблиц, многие из них станут классами-сущностями. Обозначаются классы-сущности стереотипом <<entity>> либо специальной пиктограммой (рис.6. 5).



Рисунок 6.5 - Обозначение классов-сущностей

Граничными классами называются классы, расположенные на границе системы со всем остальным миром, и т.е. они обеспечивают взаимодействие между окружающей средой и внутренними элементами системы.

Для вычисления пограничных классов необходимо исследовать диаграммы вариантов использования. Для каждого взаимодействия между актером и прецедентом нужно создать хотя бы один граничный класс. Обратите внимание, что если два действующих лица инициируют один прецедент, то они могут применять один общий пограничный класс для взаимодействия с системой. Обозначаются граничные классы именем стереотипа <<boundary>> либо специальной пиктограммой (рис. 6.6).

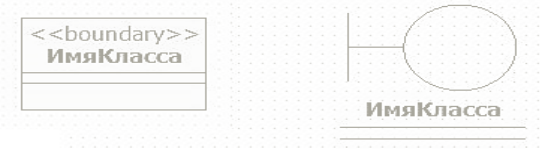


Рисунок 6 .6 – Обозначение граничных классов

Управляющий класс отвечает за координацию действий других классов. Они служат для моделирования последовательного поведения одного или нескольких прецедентов и координации событий, реализующих заложенное в них поведение. Обозначаются управляющие классы именем стереотипа <<control>> либо специальной пиктограммой (рис. 6.7).



Рисунок 6.7 – Обозначение управляющих классов

Управляющие классы можно представить, как «исполняющие» прецедент, поэтому у каждого варианта использования обычно имеется один управляющий класс, контролирующий последовательность событий этого прецедента. Они обычно зависят от приложения.

Управляющий класс делегирует ответственности другим классам. Сам он может получать мало сообщений, но отсылать множество. Его называют классом-менеджером. Он запускает альтернативные потоки и знает, как поступить в случае ошибки. На начальном этапе проектирования управляющие классы создаются для каждой пары актор/прецедент, в дальнейшем они могут объединяться, разделяться или исключаться.

После того, как класс создан, информацию о нем необходимо документировать. Заметим, что документация предназначена для описания предназначения класса, а не его структуры.

В StarUML документирование классов выполняется также, как и описанное ранее документирование прецедентов.

§ 6.2 Пакеты в языке UML

Если в разрабатываемой модели немного классов, то ими легко управлять, однако многие системы содержат большое количество классов, поэтому необходим механизм, позволяющий классы группировать и облегчающий их повторное использование. Таким механизмом в UML выступают пакеты.

Пакет (package) – общецелевой механизм для организации различных элементов модели в группы.

Подпакет (subpackage) – пакет, который является составной частью другого пакета.

Пакет в логическом представлении модели - это объединение классов или других пакетов. С помощью объединения классов в пакеты мы можем получить представление о системе на более высоком уровне. Напротив, рассматривая пакет, мы получаем более детальное представление модели.

Объединять классы в пакеты можно как угодно, однако, существует несколько наиболее распространенных подходов:

1. можно группировать классы по стереотипам: классы-сущности, граничные и управляющие классы;
2. группировка классов по их функциональности: например, пакет классов, отвечающих за безопасность системы или пакет классов Работа с сотрудниками и т.п.;
3. наконец, применяют комбинацию двух указанных методов.

В дальнейшем можно вкладывать пакеты друг в друга.

Чаще всего пакет на диаграмме изображается в виде папки с закладкой с именем пакета



Для того чтобы создать пакет на диаграмме, нужно открыть рабочее поле диаграммы, щелкнуть по элементу пакет Package на панели элементов слева, затем щелкнуть по рабочему полю диаграммы в том месте, где вы хотите поместить пакет. В окне редактора свойств можно задать новое имя пакета.

Чтобы разместить классы по пакетам, используют метод перетаскивания: на навигаторе модели нужно перетащить, удерживая левую кнопку мыши, классы в соответствующие пакеты на навигаторе модели [24].

§ 6.3 Создание диаграммы Классов

Постановка задачи: Необходимо сгруппировать в пакеты классы, созданные при выполнении предыдущих работ. Затем нужно будет построить несколько диаграмм Классов и показать на них классы и пакеты системы.

Объедините обнаруженные классы в пакеты. Создайте диаграмму Классов для отображения пакетов, диаграммы Классов, для представления классов в каждом пакете и диаграмму Классов для представления всех классов варианта использования "Ввести новый заказ".

Этапы выполнения упражнения Создание пакетов

1. В обозревателе моделей щелкнуть правой кнопкой мыши по разделу <<useCaseModel>> и выбрать Add>Package (рисунок 6.8).

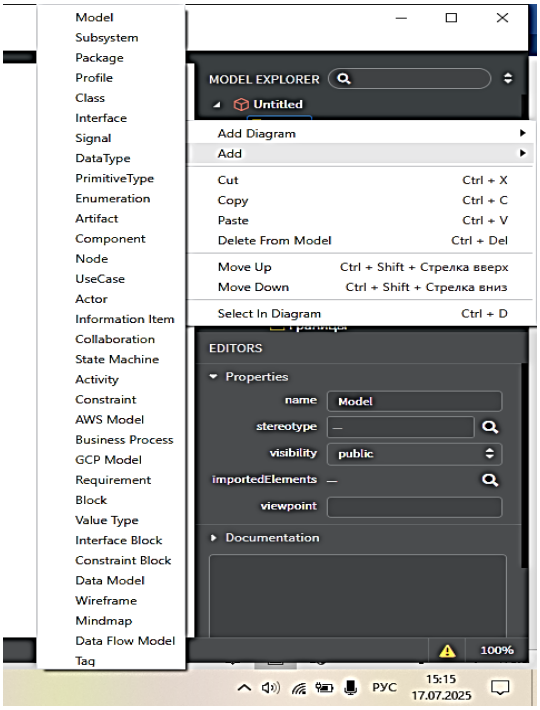


Рисунок 6.8 – Создание пакета

2. Назвать новый пакет Сущности.

3. Повторить шаги 1-2, создать пакеты Границы и Управление.

Создание Главной диаграммы Классов

20. В навигаторе моделей выбрать с помощью контекстного меню создайте новую диаграмму классов **ClassDiagram**, открыть ее.

21. Перетащить пакет *Сущности* из браузера на диаграмму.

22. Перетащить пакеты *Границы* и *Управление* из браузера на диаграмму. Главная диаграмма Классов должна выглядеть, как показано на рис. 6.9

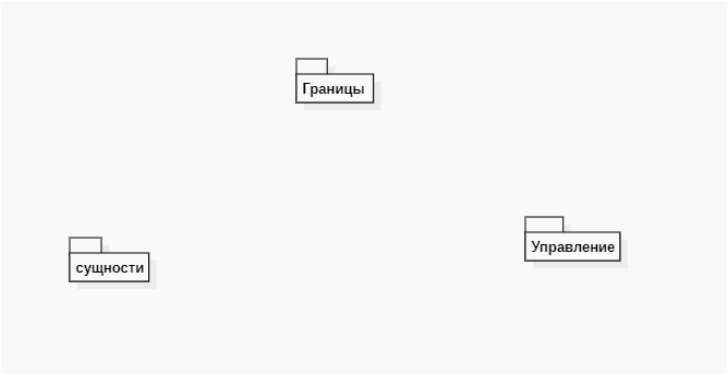


Рисунок 6. 9 – Диаграмма классов в логическом представлении браузера

Создание диаграммы Классов для сценария "Ввести новый заказ" с отображением всех классов

2. В навигаторе моделей выбрать с помощью контекстного меню создайте новую диаграмму классов **ClassDiagram**.

3. Назвать новую диаграмму Классов: Ввод нового заказа.

4. Дважды щелкнув мышью на этой диаграмме в браузере, открыть ее.

5. Перетащить из браузера все классы (*Выбор_заказа,Заказ_деталей,упр_заказами,Заказ, Упр_транзакциями*

Объединение классов в пакеты

1. В браузере перетащить класс *выбор_заказа* на пакет Границы.

2. Перетащить класс *заказ_деталей* на пакет Границы.

3. Перетащить классы *Упр_заказами* и *Упр-транзакциями* на пакет Управление.

4. Перетащить класс *Заказ* на пакет Сущности.

Классы и пакеты в браузере показаны на рис. 7.0

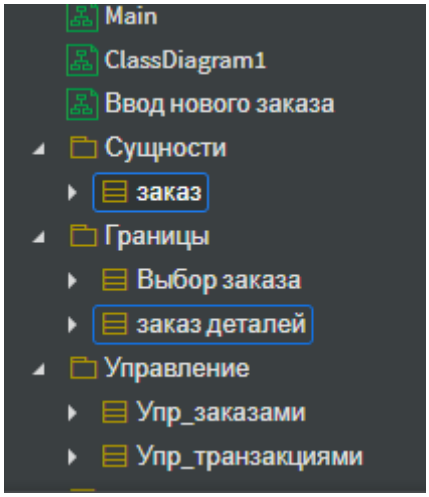


Рисунок 7.0 – Представление пакетов и классов Добавление диаграмм Классов к каждому пакету

Добавление диаграмм Классов к каждому пакету

1. В браузере щелкнуть правой кнопкой мыши на пакете *Границы*.

2. В открывшемся меню выбрать пункт AddDiagram>ClassDiagram.

3. Ввести имя новой диаграммы – Main (Главная).

4. Перетащить на нее из браузера классы *выбор_заказа* и *заказ_деталей*.

5. В браузере щелкнуть правой кнопкой мыши на пакете *Сущности.*

8. В открывшемся меню выбрать пункт AddDiagram>ClassDiagram.

9. Ввести имя новой диаграммы – Main (Главная).

10. Перетащить на нее из браузера класс *Заказ*.

11. В браузере щелкнуть правой кнопкой мыши на пакете *Управление*

12. В открывшемся меню выбрать пункт AddDiagram>ClassDiagram.

13. Ввести имя новой диаграммы – Main (Главная).

14. Перетащить на нее из браузера классы *Упр_заказами* и *Упр_транзакциями[24]*

§ 6.4 Уточнение методов и свойств классов

К стереотипным операциям для всех классов относятся следующие операции: − создать и инициировать (create and initialize) объект;

− удалить (delete) объект. − получить (get…) значение атрибута;

− установить (set…) значение атрибута;

− добавить объект (add…);

– добавить связь с другим объектом;

− исключить объект (remove…) – исключить связь с другим объектом.

К наиболее распространённым операциям относятся:

− модификатор – операция, которая меняет состояние объекта;

− запрос (селектор) – операция, которая считывает состояние объекта, но не меняет состояние;

− итератор – операция, которая позволяет организовать доступ ко всем частям объекта-контейнера для последующего их использования в строго определённой последовательности (например, просмотр очереди);

− преобразование – операция, производящая объект другого типа.

Метод – это конкретная реализация операции.

Постановка проблемы

Для определения атрибутов классов был проанализирован поток событий. В результате к классу Заказ диаграммы Классов были добавлены атрибуты Номер заказа и Имя клиента. Так как в одном заказе можно указать большое количество товаров и у каждого из них имеются свои собственные данные и поведение, было решено моделировать товары как самостоятельные классы, а не как атрибуты класса Заказ.

Добавление атрибутов и операций

Добавим атрибуты и операции к классам диаграммы Классов «Ввод нового заказа». При этом используем специфические для языка особенности. Установим параметры так, чтобы показывать все атрибуты, все операции и их сигнатуры. Применим нотацию UML.

Этапы выполнения упражнения

Настройка

1. В меню модели выбрать пункт Format.

2. Убедиться, что флажок Show visibility (Показать видимость) установлен.

3. Убедиться, что флажок Show operation signatures (Показать сигнатуры операций) установлен.

4. Убедиться, что флажки Suppress attributes (Подавить атрибуты) и Suppress operations (Подавить операции) сброшены.

5. Перейти на вкладку Stereotype Display.

6. Убедиться, что флажок Icons (Отображать пиктограммы) сброшен.

Добавление нового класса

1. Найти в браузере диаграмму Классов варианта использования «Ввод нового заказа».

2. Дважды щелкнуть мышью на диаграмме, откройте ее.

3. Щелкнуть кнопку Entity в TOOLBOX.

4. Щелкнуть мышью внутри диаграммы, чтобы поместить туда новый класс.

5. Назвать его *Позиц_заказа* (рисунок 7.1).

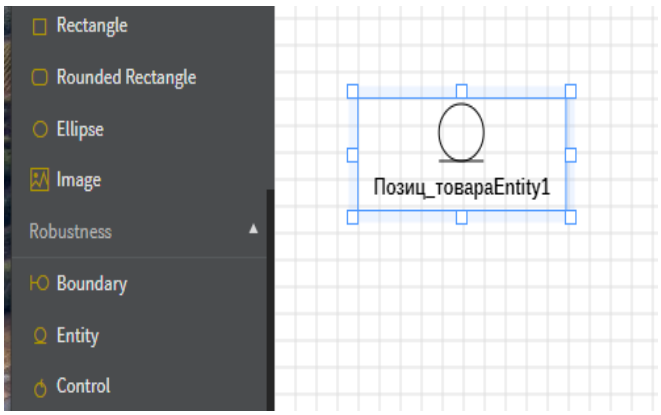


Рисунок 7.1 – Назначение классу Позиция заказа стереотипа Entity

6. В браузере перетащить класс в пакет *Сущности.*

7. Стереотипы классов можно отобразить с помощью пиктограмм. Для этого нужно выделить класс, щелкнуть по выделенной области правой кнопкой мыши, в контекстном меню выберите пункт Format, затем выбрать пункт Stereotype Display, далее в списке выбрать Iconic (рисунок 7.2).

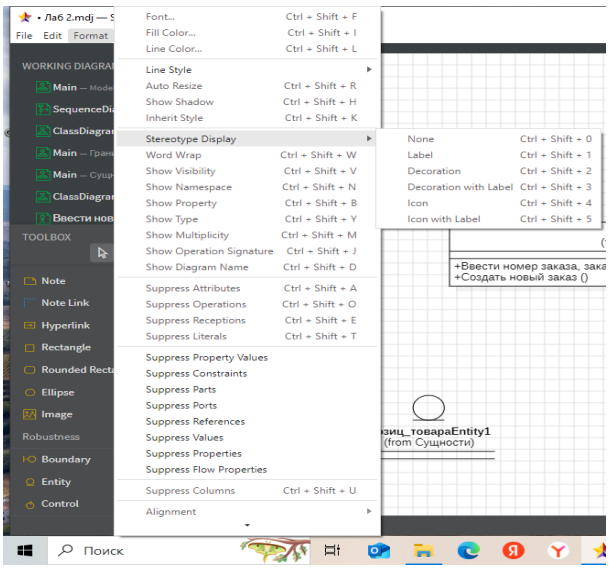


Рисунок 7.2 – Задание отображения стереотипов классов с помощью пиктограмм

Классы будут отображаться как пиктограммы (рисунок 7.3)

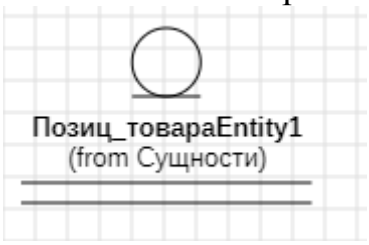


Рисунок 7.3 – Отображение классов с помощью пиктограмм стереотипов

Добавление атрибутов

1. Щелкнуть правой кнопкой мыши на классе *Заказ*.
2. В открывшемся меню выбрать пункт Add Attribute (Создать атрибут),
3. Ввести новый атрибут: **OrderNumber**:Integer
4. Нажать клавишу Enter
5. Ввести следующий атрибут: **CustomerName**:String.
5. Повторив шаги 4 и 5, добавить атрибуты: **OrderDate**:Date, **OrderFillDate**:Date

Если тип атрибута не появляется в выпадающем списке, то ввести его от руки и он далее будет появляться.

7. Щелкнуть правой кнопкой мыши на классе *Позиц_заказа*.
8. В открывшемся меню выбрать пункт NewAttribute (Создать атрибут).
9. Ввести новый атрибут: ***ItemID*:Integer.**
10. Нажать клавишу Enter.
11. Ввести следующий атрибут: ***ItemDescription*:String.**

Добавление операций к классу *Позиц_заказа*

1. Щелкнуть два раза на классе *Позиц_заказа*.
2. В открывшемся меню выбрать пункт Add Operation (рисунок 7.5).
3. Ввести новую операцию: *Создать()*

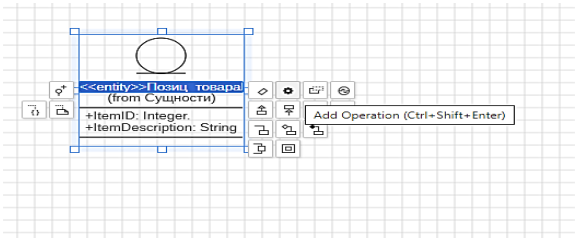


Рисунок 7.5 –Добавление операции к классу

4. Нажать клавишу Enter.
5. Ввести следующую **операцию:** *Взять_информацию()*
6. Нажать клавишу Enter.
7. Ввести операцию: *Дать_информацию()*

Подробное описание операций с помощью диаграммы Классов

1. Щелкнуть мышью на классе *Заказ*, выделите его.

2. Щелкнуть на этом классе еще раз, чтобы переместить курсор внутрь.

3. Отредактировать операцию *Создать()*, чтобы она выглядела следующим образом:

Создать (): Boolean

4. Отредактировать операцию *Взять_информацию*:

Взять_информацию(OrderNum: Integer, Customer: String, OrderDate: Date,FillDate: Date): Boolean

5. Отредактировать операцию

Дать_информацию; *Дать_информацию()***: String**

Подробное описание операций с помощью браузера

1. Найти в браузере класс *Позиц_заказа*.

2. Раскрыть этот класс, щелкнув на знак рядом с ним. В браузере появятся атрибуты и операции класса (рисунок 7.6).

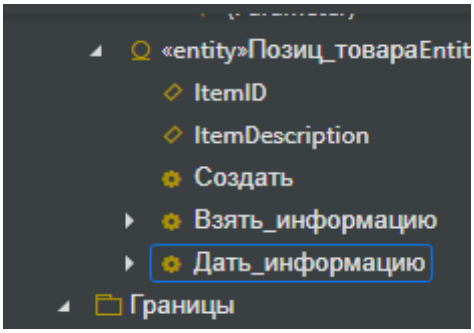


Рисунок 7.6 – Определение типа атрибута

3. Кликнуть мышью на операции *Дать_информацию()* , открытье окно свойств Properties

7. В свойстве Type написать тип String .

4. Еще раз кликнуть мышью на операции *Дать_информацию()* , открыть окно свойств Properties

8. В свойстве Type изменить тип String на Boolean (рисунок 7.7).

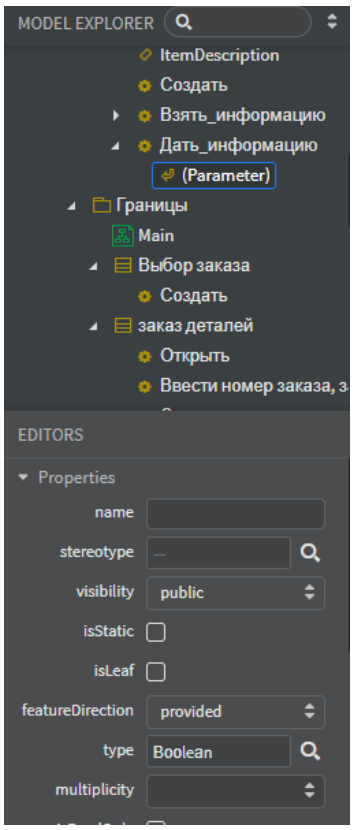


Рисунок 7.7- Определение типа атрибута

9. Кликнуть правой кнопкой мыши на операции Дать_информацию()

10.В открывшемся меню выбрать пункт Add Parameter (рис.7.8), чтобы добавить туда новый параметр

11. Щелкнув один раз на этом слове, выделить его и изменить имя аргумента на ID.

12. Щелкнуть на колонке Type (Тип).

13.В раскрывающемся списке типов выбрать Integer (Если этого либо иного необходимого типа не будет введите его вручную).

14.Щелкнуть на колонке Default (По умолчанию), чтобы добавить значение аргумента по умолчанию. Ввести число 0.

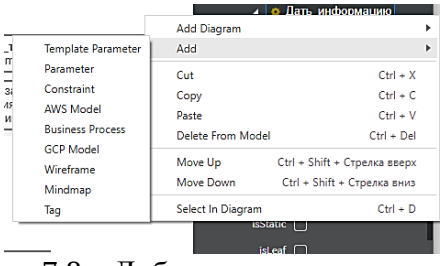


Рисунок 7.8 – Добавление нового параметра

15.Кликнуть правой кнопкой мыши на операции Создать() класса Позиц_заказа,

16. Добавить новый параметр

17.В свойствах указать тип Boo1ean

Подробное описание операций

1. Используя браузер или диаграмму Классов, ввести следующие сигнатуры операций класса *Заказ_деталей*: *Открыть()*: Boolean, *Сохранить заказ()*: Boolean

2. Используя браузер или диаграмму Классов, введите ввести сигнатуру операций класса *Выбор_заказа*: *Создать()* :Boolean

3. Используя браузер или диаграмму Классов, ввести сигнатуру операций класса *Упр_заказами*: *Сохранить заказ*(OrderID :Integer) :Boolean

4. Используя браузер или диаграмму Классов, ввести сигнатуры операций класса *Упр_транзакциями*: *Сохранить заказ* (OrderID:Integer): Boolean, *Сохранить информацию*():Integer (рис.7.9).

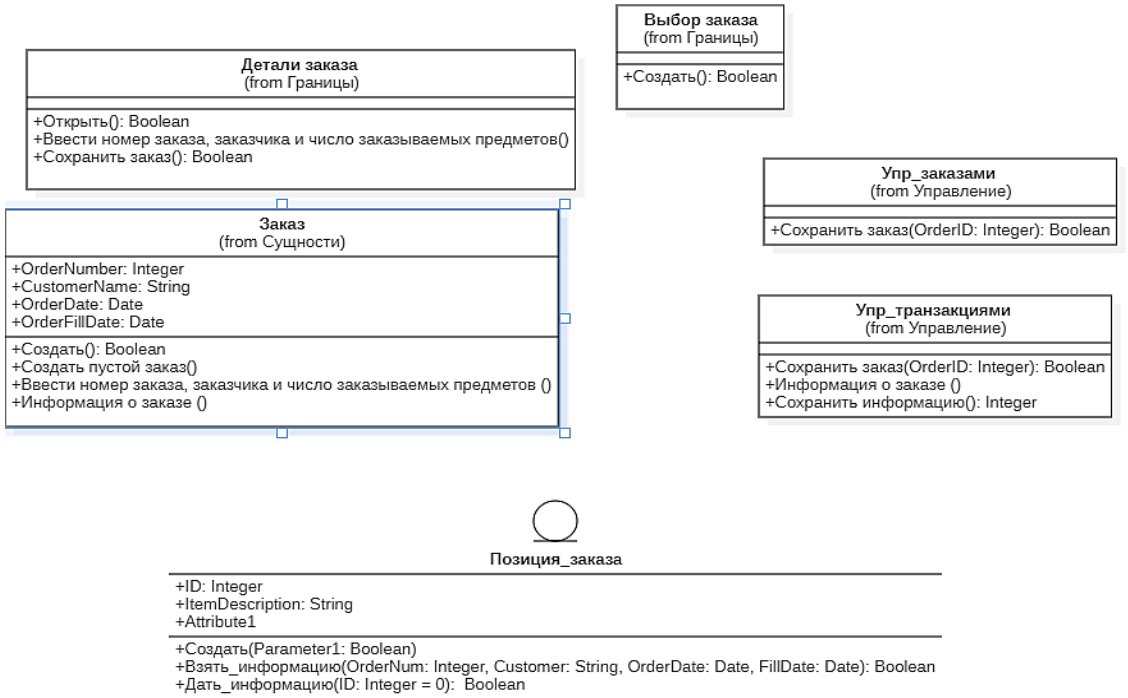


Рисунок 7.9

§ 6.5 Типы связей между классами UML

Выделяется шесть основных типов отношений между классами: наследование, реализация, композиция, агрегация, ассоциация и зависимость. Условные обозначения этих отношений представлены ниже (рис.8.0)

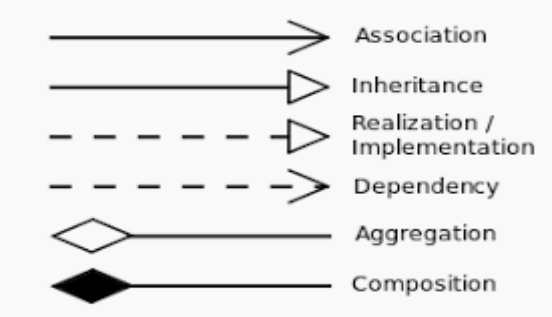


Рисунок 8.0

Наследование (обобщение) используется для описания взаимосвязи между родительскими и дочерними классами. Родительский класс называется базовым классом, а подкласс – производным классом.

В отношениях наследования подкласс наследует все функции родительского класса, а родительский класс характеризуется всеми атрибутами, методами и подклассами. Подклассы содержат дополнительную информацию в дополнение к той же информации, что и родительский класс.

Например: автобусы, такси и автомобили – это автомобили, у всех них есть имена, и все они могут находиться на дороге.

Реализация в основном используется для определения взаимосвязи между интерфейсами и классами реализации.

Интерфейс (включая абстрактный класс) – это набор методов. В отношениях реализации класс реализует интерфейс, а методы в классе реализуют все методы, объявленные в интерфейсе.

Например: автомобили и корабли – это транспортные средства, причем транспортное средство – это просто абстрактное понятие мобильного транспортного средства, а корабль и транспортное средство выполняют определенные мобильные функции.

Ассоциация как наиболее часто используемая связь между классами, означает наличие связи между одним типом объекта и другим типом объекта.

Например: менеджер может выписать счет – возникает ассоциация между менеджером и счетом или преподаватель и студент – студен учится у преподавателя.

Выделяют четыре вида ассоциации: односторонние, двусторонние, самоссоциация и многозначные ассоциации.

В отношении множественности вы можете добавить число непосредственно к связанной строке, чтобы указать количество объектов в соответствующем классе.

1..1: Единственный

0..*: ноль или больше

1..*:один или больше

0..1: Нет или только один

m..n: не менее m, не более n (m<=n)

Рассмотрим каждый тип кратности на примерах.

1..1: Единственный – означает ровно один. Один человек имеет ровно один действующий паспорт, со стороны паспорта: один паспорт принадлежит ровно одному человеку.

0..*: Ноль или больше – означает возможно отсутствие связи, а может быть и любое количество объектов. В качестве примера рассмотрим Человека и его твиты (посты в соцсети). Один человек может не иметь ни одного твита (0), а может написать их множество (*).

1..*: Один или больше-означает как минимум один, но может быть и больше, т.е. должно быть обязательно один. В качестве примера можно рассмотреть заказ в интернет-магазине и позицию заказа, т.е. любой оформленный заказ должен содержать как минимум одну товарную позицию,

а обычно несколько.

0..1: Ноль или один- означает либо связи нет, либо она есть ровно с одним объектом. Сотрудник компании и номер его рабочего телефона. В кабинете сотрудника есть внутренний телефон или его нет.

m..n:от mдо n (m<=n) –означает конкретный диапазон, например2..4,1..5, 0…18. Количество объектов должно находиться в этом диапазоне (включительно).

Связь обычно имеет две кратности – по одному числу на каждом конце линии, обозначающей ассоциацию.

Например:

[сотрудник]1_____0..1 [рабочий телефон]

Рядом с сотрудником стоит 1: один рабочий телефон закреплен за сотрудником.

Рядом с рабочим телефоном стоит 0..1: у одного конкретного сотрудника есть это рабочий номер (1), либо у него нет рабочего телефона (0) – это значит что этот номер может быть свободен.

Агрегация – это отношения между целым и частью, при этом целое и часть могут быть разделены, т.е. объекты-члены являются частью общего объекта, но объект-член может существовать независимо от общего объекта.

Например, водители автобусов, рабочая одежда и головные уборы являются частью общих отношений, но их можно разделить. Другие водители могут носить эту рабочую одежду и головные уборы. Водители автобусов также могут носить другую рабочую одежду и другие головные уборы(рис.8.1).

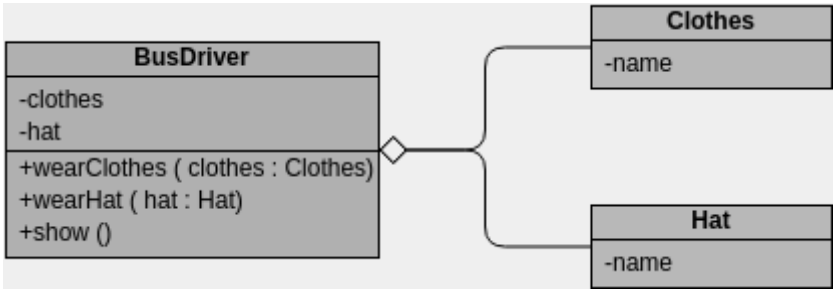


Рисунок 8.1

Композиция рассматривается как отношения между целым и частью, но целое и часть не могут существовать без друг друга. Это частный случай агрегации, при котором составные части целого имеют такое же время жизни, что и само целое, и уничтожаются вместе с уничтожением целого.

При моделировании отношений между рабочими группами и сотрудниками в информационной системе отдела кадров используется композиция. Если в организации принята жесткая структура: каждый сотрудник входит ровно в одну рабочую группу и в каждой рабочей группе есть по меньшей мере один сотрудник.

Зависимость отражает связь между объектами по применению, когда изменение поведения одного класса может повлиять на поведение другого. В

большинстве случаев зависимости отражаются в методах класса, использующих в качестве параметра объект другого класса. Например: магазин делает заказ на склад. Выполнение заказа зависит от того, есть ли на складе то или иной продукт и его количество. Метод объекта «Магазин» должен использовать объект класса «Склад», чтобы узнать наличие на складе тех или иных продуктов.

(Class Diagram Tutorial https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/)

В этом упражнении определяются связи между классами, участвующими в варианте использования «Ввести новый заказ».

Постановка задачи

Чтобы найти связи, были просмотрены диаграммы Последовательности. Все взаимодействующие там классы нуждались в определении соответствующих связей на диаграммах Классов. После обнаружения связи были добавлены на диаграммы классов.

Добавление связей

Добавим связи к классам, принимающим участие в варианте использования "Ввести новый заказ".

Этапы выполнения упражнения

Добавление ассоциаций

1. Нажать кнопку Directed Association панели инструментов.

2. Провести ассоциацию от класса *выбор_заказа* к классу *заказ_деталей*.

3. Повторить шаги 1 и 2, создав ассоциации:

- От класса *заказ_деталей* к классу *упр_заказами*
- От класса *упр_заказами* к классу *Заказ*
- От класса *упр_заказами* к классу *упр_транзакциями*
- От класса *упр_транзакциями* к классу Заказ
- От класса *упр_транзакциями* к классу *Позиц_заказа*
- От класса *Заказ* к классу *Позиц_заказа*

4. указать два раза на однонаправленной ассоциации между классами *выбор_заказа* и *заказ_деталей* класса *выбор_заказа*.

Задать значение 0..1 пользуясь свойствами End1.Multiplicity окна Properties (рис.8.2)

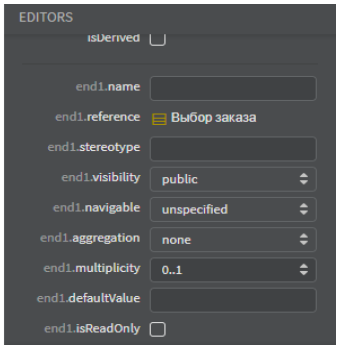


Рисунок 8.2 – Задание свойства множественности в окне Properties

5. В открывшемся меню выбрать пункт Multiplicity> (Множественность > Нуль или один),

6. Повторить шаги 4-7, добавив на диаграмму значения множественности для остальных ассоциаций, как показано на рис. 8.3

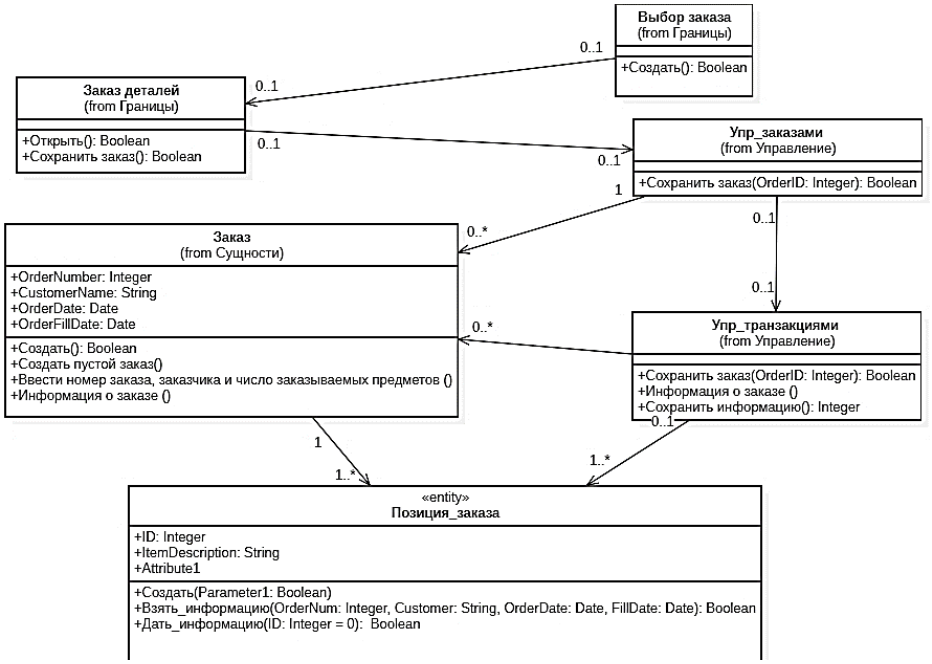


Рисунок 8.3 – Ассоциации сценария "Ввести новый заказ"

Задание: Диаграмму классов сценария Оформление заказа отобразить с помощью пиктограмм стереотипов.

Для создания новой диаграммы классов выполним следующие шаги: щелкнуть правой кнопкой мыши по папке представления Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, в списке выбрать диаграмму классов Class Diagram. Будет создана новая диаграмма классов со стандартным именем ClassDiagram1, которое можно изменить в редакторе свойств диаграммы.

Рассмотрим сценарий Оформление заказа, который является внутренним потоком для прецедента Заказ товара. Опишем его классы.

Данный сценарий позволяет покупателю оформить заказ из корзины и оплатить его с использованием банковской карты.

Рассмотрим как выполняется этот сценарий, еще раз. Покупатель, находясь в своей покупательской корзине, приняв решение о том, что он готов сделать заказ в магазине «Style», выбирает опцию «Оформить заказ». Как реагирует система на действия покупателя? Запускается сценарий Оформление заказа. Пользователь должен на специальной форме внести свои личные данные, подтвердить заказ или нет, и в зависимости от этого произвести оплату, затем получить подтверждение заказа. В системе появляется новый объект – **заказ покупателя.**

Выбор граничных классов.

По всей видимости, нам нужен будет хотя бы один граничный класс, который осуществляет связь между действующим лицом Покупатель и дополнительным прецедентом Оформить заказ. Назовем его **Оформление Заказа (PlaceOrder**). Этот класс знает, какие товары и в каком количестве были в корзине покупателя, их нужно перенести в заказ. Также этот класс может знать, пуста корзина покупателя или нет, и, если пуста, то вывести об этом соответствующее сообщение.

Для того, чтобы сделать заказ, покупатель должен ввести свои личные данные, электронный адрес, телефон и данные кредитной карты. Для этих целей мы введем еще один класс Ввод Личных Данных (Enter Personal Information). После того, как выбраны товары и введена личная информация покупателя, остается только проверить детали заказа и согласиться с ними или нет – для этого действия введем класс **Проверка Деталей Заказа (ConfirmOrder**). Наконец, когда покупатель завершит оформление заказа, то на экране он видит номер заказа и подтверждение заказа отправляется покупателю на электронный адрес, для выполнения этих обязанностей создадим еще один граничный класс **Подтверждение Заказа (Order Confirmation)[9]**

Выбор управляющих классов.

Создадим один управляющий класс, который будет распределять обязанности других классов и вызывать их операции при выполнении данного сценария. Назовем этот управляющий класс **Менеджер Оформления Заказа (Place Order Manager).**

Выбор классов-сущностей.

В сценарии Оформление заказа речь идет о покупателе, заказе и товарах. Создадим классы-сущности **Покупатель (Customer), Заказ (Order), Товар (Item) (рис.8.4-8.6).**

Возможно, что в ходе дальнейшего проектирования какие-то новые классы будут добавлены для этого сценария, а какие-то, напротив, из этой диаграммы удалены.

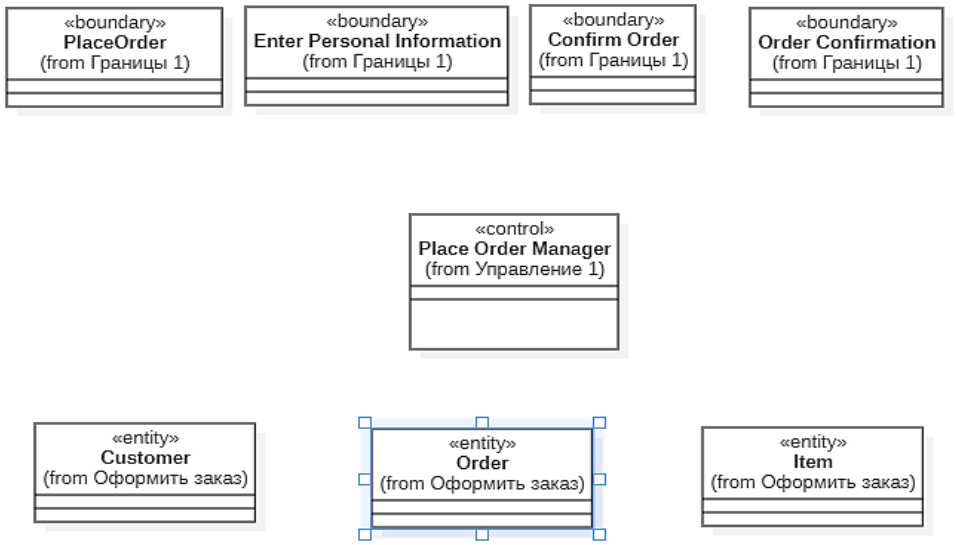


Рисунок 8.4

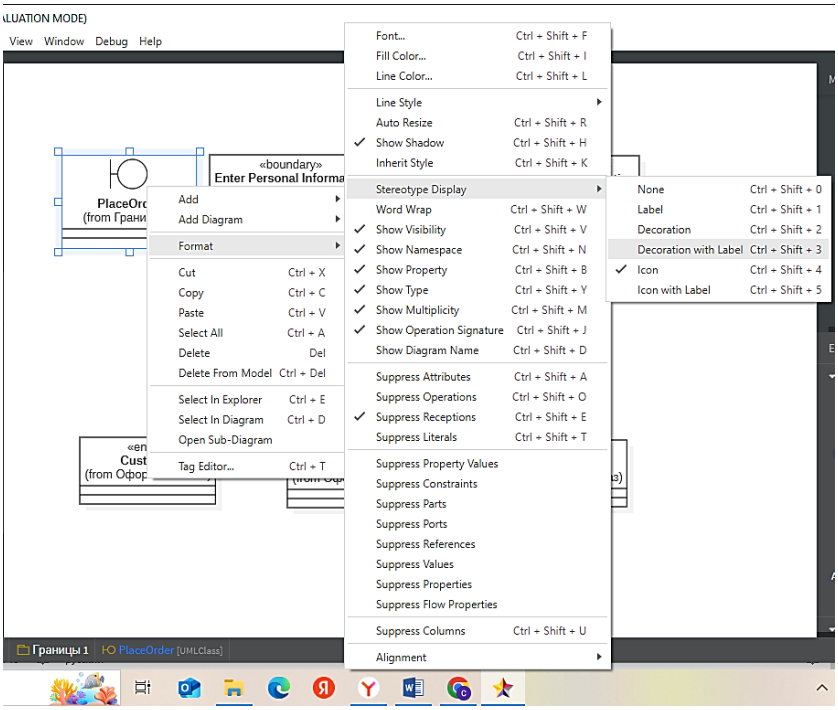


Рисунок 8.5

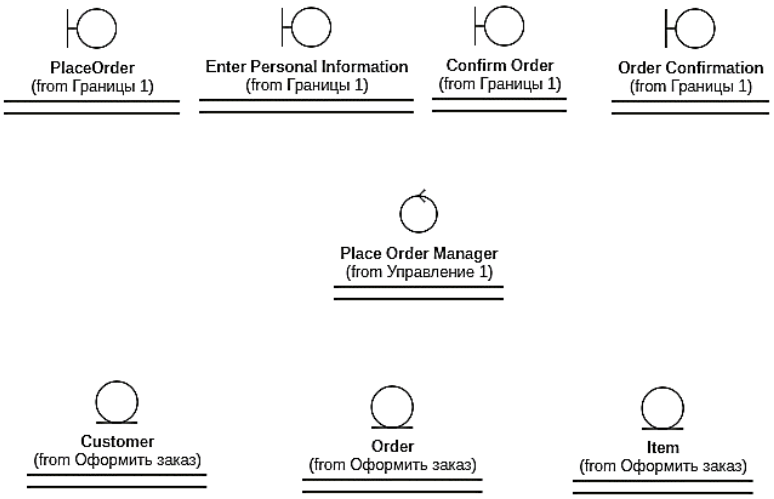


Рисунок 8.6

Вопросы для самоконтроля

1. Какие элементы может содержать диаграмма классов?
2. Какие свойства спецификации имеет класс?
3. Что называют сообщением в UML?
4. Какого типа отношения существуют между классами?
5. В каком случае устанавливается связь агрегации? Как определяется ее направленность?
6. Чем отличается отношение композиции от отношения агрегации?
7. В каком случае устанавливается связь обобщения? Как определяется ее направленность?
8. Каким образом может быть использована диаграмма классов?
9. В каком случае необходимо именовать отношение ассоциации или ее полюсов?
10. Прообразом чего в программном коде является операция класса?
11. Что такое интерфейс? В каких случаях в диаграмму классов включают интерфейсы?
12. Как определить, является ли класс классом сущности?

§ 7. Диаграмма компонентов и особенности ее построения

В отличие от ранее рассмотренных диаграмм, диаграмма компонентов описывает свойства физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, устанавливая связи между компонентами программы, как исходный код,

бинарные файлы и исполняемые файлы. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают взаимозависимые отношения, аналогичные тем, которые возникают при компиляции исходного кода программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и отношения между ними [24].

В разработке диаграммы компонентов участвуют системные аналитики, архитекторы и программисты. Диаграмма компонентов обеспечивает согласованный переход от логического представления для реализации конкретного проекта в виде программного кода. Некоторые компоненты могут существовать только на этапе компиляции программного кода, в то время как другие могут существовать на этапе выполнения. Компонентная диаграмма показывает общую взаимосвязь между компонентами, рассматривая последние как отношение между ними.

В языке UML для обозначения физических сущностей используется специальный термин «компонент».

Компонент – это физически существующая часть системы, которая обеспечивает реализацию классов и отношений моделируемой программной системы, а также её функциональное поведение.

Компонент выступает в качестве общего обозначения элементов физического представления модели и может реализовывать определенные интерфейсы. Для графического представления компонента используется специальный прямоугольный символ с двумя маленькими прямоугольниками слева (рис. 7.1). Имя компонента и, возможно, дополнительная информация записываются внутри прямоугольника. Этот символ является основным обозначением компонента в языке UML.

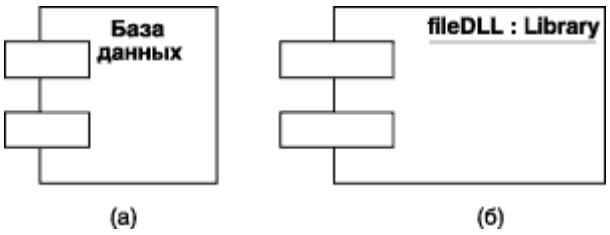


Рисунок 7.1 – Графическое изображение

Имя компонента соответствует общим правилам именования элементов модели UML и может содержать любое количество букв, цифр и знаков препинания. Один компонент может быть представлен на уровне типа или экземпляра. Хотя графическое представление одинаково в обоих случаях, правила написания имени компонента немного различаются.

Если компонент представлен на уровне типа, только имя типа пишется с заглавной буквы в форме: <Имя типа>. Если компонент представлен на уровне экземпляра, его имя записывается в формате: <имя компонента: 'имя типа>. В этом случае подчеркивается всё имя целиком. В первом случае (рис.7.1, а) для компонента уровня типа указывается имя типа, а во втором

(рис. 7.1, б) для компонента уровня экземпляра указывается соответствующее имя компонента и имя типа.

Правила именования объектов UML требуют выделения имен отдельных экземпляров, но, когда речь идет о компонентах, их имена часто опускаются. В этом случае написание имени компонента строчными буквами характеризует компонент уровня примера.

В качестве подходящих имен компонентов обычно используются имена исполняемых файлов, динамических библиотек, веб-страниц, текстовых или справочных файлов, файлов баз данных или скриптовых файлов, содержащих исходный код программ, и другие.

Поскольку компонент как элемент модели может иметь различные физические применения, иногда он изображается в виде специального графического символа, отражающего специфические характеристики применения. Строго говоря, эти дополнительные обозначения не указаны в нотации языка UML. Однако, удовлетворяя общие механизмы расширения для языка UML, они значительно повышают наглядность графического представления, упрощая понимание диаграммы компонентов.

Для более наглядного изображения компонентов были предложены и стали общепринятыми следующие графические стереотипы:

- Во-первых, стереотипы для компонентов развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки (рис.7.1, б), Web-страницы на языке разметки гипертекста (рис.7.2, б) и файлы справки (рис.7.2, в).
- Во-вторых, стереотипы для компонентов в форме рабочих продуктов. Как правило – это файлы с исходными текстами программ (рис.7.2, г).

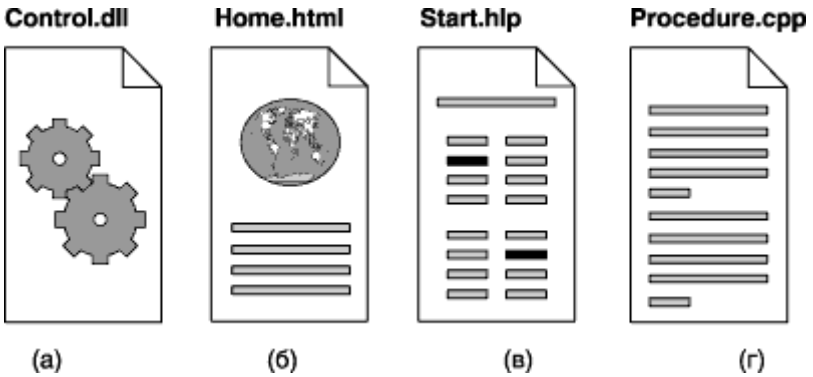


Рисунок 7.2 – Варианты графического изображения компонентов на диаграмме компонентов.

Эти элементы называют артефактами, хотя их полное содержание включает информацию, а соответствующие компоненты реализуются с использованием специальной технологии. Кроме того, разработчики могут использовать для этой цели свои обозначения, поскольку в языке UML нет строгой нотации для графического представления артефактов.

Другой способ спецификации различных типов компонентов - указание текстового стереотипа компонента перед его именем. Ниже приведены некоторые стереотипы языка компонентов UML:

<<file>> (файл) – определяет наиболее распространенный тип компонента, представленного в виде произвольного физического файла.

<<executable>> (исполнимый) – определяет тип компонента- файла, который является исполнимым файлом и может быть выполнен на компьютерной платформе.

<<document>> (документ) – определяет тип компонента-файла, представленного в виде документа произвольного содержимого, который не является исполнимым файлом или файлом с исходным кодом программы.

<<library>> (библиотека) – определяет тип компонента- файла, представленного в виде динамической или статической библиотеки.

<<source>> (источник) – определяет тип компонента-файла, который представляет собой файл с исходным кодом программы, который после компиляции может быть преобразован в исполнимый файл.

<<table>> (таблица) – определяет тип компонента, представленного в виде таблицы база данных.

Отдельные разработчики предложили свои собственные графические стереотипы для описания определенных типов компонентов, но они не получили широкого распространения, за некоторыми исключениями.

Интерфейсы

Следующим графическим элементом диаграммы компонентов являются интерфейсы. Обычно интерфейс графически представляется кругом, соединяющим компонент отрезком без стрелки (рис. 7.3, а). В этом случае рядом с кругом пишется название интерфейса, которое рекомендуется начинать с буквы «I». Семантически линия означает реализацию интерфейса, а наличие интерфейсов в компоненте означает, что этот компонент реализует соответствующий набор интерфейсов.

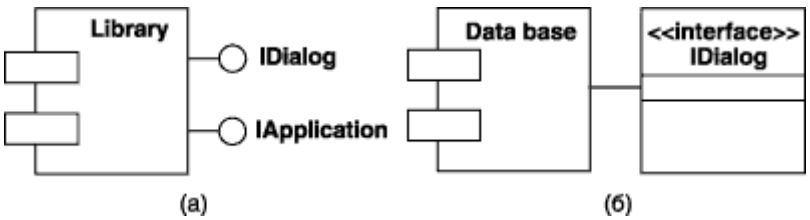


Рисунок 7.3 – Графическое изображение интерфейсов на диаграмме компонентов.

Интерфейс также может быть представлен на диаграмме компонентов в виде прямоугольника класса со стереотипом <<Интерфейс>> и разделом поддерживаемых операций (рис. 7.3, б). Обычно это используется для представления внутренней структуры интерфейса.

При разработке программных систем интерфейсы обеспечивают не только совместимость различных версий, но и возможность внесения существенных изменений в одни части программы без изменения других. Реализация интерфейсов отдельными компонентами может различаться.

Существует два способа взаимодействия между интерфейсом и компонентом. Если компонент реализует определенный интерфейс, этот интерфейс называется экспортируемым или поддерживаемым, поскольку этот компонент предоставляет его в качестве услуги другим компонентам.

Если компонент использует некоторые интерфейсы, реализованные другим компонентом, такой интерфейс для первого компонента называется импортируемым.

Особенность импортируемого интерфейса состоит в том, что на диаграмме компонентов это отношение изображается с помощью зависимости.

В общем случае отношение зависимости также было рассмотрено ранее. Отношение зависимости служит для представления факта наличия специальной формы связи между двумя элементами модели, когда изменение одного элемента модели оказывает влияние или приводит к изменению другого элемента модели. Отношение зависимости на диаграмме компонентов изображается пунктирной линией со стрелкой, направленной от клиента или зависимого элемента к источнику или независимому элементу модели.

Зависимости могут отражать связи отдельных файлов программной системы на этапе компиляции и генерации объектного кода. В других случаях зависимость может указывать на наличие в независимом компоненте описаний классов, которые используются в зависимом компоненте для создания соответствующих объектов. Применительно к диаграмме компонентов зависимости могут связывать компоненты и импортируемые этим компонентом интерфейсы, а также различные виды компонентов между собой.

В этом случае от клиентского компонента к импортированному интерфейсу проведена стрелка. Наличие такой стрелки означает, что компонент не реализует соответствующий интерфейс, но использует его при его реализации. При этом на той же диаграмме может быть другой компонент, реализующий этот интерфейс. Коэффициент работы интерфейса отображается на диаграмме компонентов в виде обычной линии без стрелки.

Другой пример отношения зависимости на диаграмме компонентов - это отношения программного вызова и компиляции между различными видами компонентов.

Диаграмма компонентов также может отображать зависимости между компонентами и классами, которые в них реализованы. Эта информация важна для обеспечения согласованности между логическим и физическим описанием модели системы. Конечно, изменения в структуре описаний классов могут привести к изменению этой зависимости.

Если необходимо подчеркнуть, что компонент реализует отдельные классы, для его идентификации используется расширенный символ прямоугольника. В этом случае прямоугольник компонента разделен горизонтальной линией на две части. Верхняя часть используется для записи

имени компонента и, возможно, дополнительной информации, а нижняя часть – для отображения классов, реализованных данным компонентом

Если компонент является экземпляром и реализует три отдельных объекта, он описывается как компонент уровня экземпляра. Объекты, расположенные в отдельном компоненте-экземпляре, описываются так, как они включены в символ этого компонента. Это означает, что реализация компонента приводит к операциям с соответствующими объектами. При этом наличие компонента во время выполнения программы обеспечивает функциональность всех содержащихся в нем объектов.

Рекомендации по построению диаграммы компонентов

Разработка диаграммы компонентов предполагает использование информации не только о логическом представлении модели системы, но и о характеристиках её физического функционирования. В первую очередь необходимо определить, какие физические части или файлы будет содержать программная система. На этом этапе следует уделить внимание реализации системы, которая обеспечит повторное использование кода за счёт декомпозиции компонентов, а также созданию объектов только при необходимости.

Общая производительность программной системы во многом зависит от эффективного использования вычислительных ресурсов. С этой целью большую часть описаний классов, их операций и методов следует перенести в динамические библиотеки, оставив в исполняемых компонентах только те фрагменты программного кода, которые наиболее необходимы для работы программы.

После общей структуры физического представления системы необходимо дополнить модель интерфейсами и диаграммами баз данных. При проектировании интерфейсов следует обратить внимание на координацию различных частей программной системы. Включение базы данных в схематическую модель включает в себя спецификацию отдельных таблиц и установление связей между ними.

Заключительный этап построения диаграммы компонентов связан с установлением и нанесением связей между компонентами, а также отношений реализации. Эти отношения должны описывать все важные аспекты физической реализации системы, начиная с деталей компиляции исходного кода программ и заканчивая выполнением отдельных частей программы на этапе её реализации. Для этого можно использовать различные стереотипы графических компонентов.

При построении диаграммы компонентов следует придерживаться общих принципов построения моделей на языке UML. В частности, прежде всего необходимо использовать графические и текстовые стереотипы, существующие в языке UML и общепринятые. В большинстве типичных проектов этого набора достаточно для представления компонентов и зависимостей между ними.

Если проект содержит физические элементы, не описанные на языке UML, необходимо использовать механизм расширения. В частности, к

отдельным нетипичным компонентам можно применять дополнительные стереотипы или помеченные значения меткам для уточнения индивидуальных свойств компонентов.

§ 8. Диаграмма развертывания, особенности ее построения
§ 8.1 Диаграмма развертывания

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Если создается простая программа, которая может выполняться локально на компьютере пользователя, не используя никаких распределенных устройств и сетевых ресурсов, то необходимости в разработке дополнительных диаграмм нет. Однако при создании корпоративных или распределенных приложений требуется визуализировать сетевую инфраструктуру программной системы.

«Сложные программные системы могут реализовываться в сетевом варианте, на различных вычислительных платформах и технологиях доступа к распределенным базам данных. Наличие локальной корпоративной сети требует решения целого комплекса дополнительных задач рационального размещения компонентов по узлам этой сети, что определяет общую производительность программной системы.

Интеграция программной системы с Интернетом определяет необходимость решения дополнительных вопросов при проектировании системы, таких как обеспечение безопасности и устойчивости доступа к информации для корпоративных клиентов. Эти аспекты в немалой степени зависят от реализации проекта в форме физически существующих узлов системы, таких как серверы, рабочие станции, брандмауэры, каналы связи и хранилища данных"[12, с.320].

Технологии доступа и манипулирования данными в рамках общей схемы «клиент-сервер» также требуют размещения больших баз данных в различных сегментах корпоративной сети, их резервного копирования, архивирования, кэширования для обеспечения необходимой производительности системы в целом. С целью спецификации программных и технологических особенностей реализации распределенных архитектур необходимо визуальное представление этих аспектов.

Первой из диаграмм физического представления является диаграмма компонентов. Вторая форма физического представления программной системы - это диаграмма развертывания (размещения).

Диаграмма развертывания (deployment diagram) – диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов.

Диаграмма развертывания применяется для представления общей конфигурации и топологии распределенной программной системы и содержит изображение размещения компонентов по отдельным узлам

системы. Кроме того, диаграмма развертывания показывает наличие физических соединений – маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих только на этапе ее исполнения (run-time). При этом представляются только те компоненты программы, которые являются исполнимыми файлами или динамическими библиотеками. Компоненты, не используемые на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единственной для системы в целом, поскольку должна отражать все особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, последний этап спецификации модели. Диаграмма развертывания разрабатывается совместно системными аналитиками, сетевыми инженерами.

Узел (node) представляет собой физически существующий элемент системы, который может обладать вычислительным ресурсом или являться техническим устройством[12].

В качестве вычислительного ресурса узла может рассматриваться один или несколько процессоров, а также объем электронной или магнитооптической памяти. Однако в языке UML понятие узла включает в себя не только вычислительные устройства (процессоры), но и другие механические или электронные устройства, такие как датчики, принтеры, модемы, цифровые камеры, сканеры и манипуляторы.

Графически узел на диаграмме развертывания изображается в форме трехмерного куба. Узел имеет имя, которое указывается внутри этого графического символа. Сами узлы могут представляться как на уровне типа (рис.8.1, а), так и на уровне экземпляра (рис. 8.1, б).

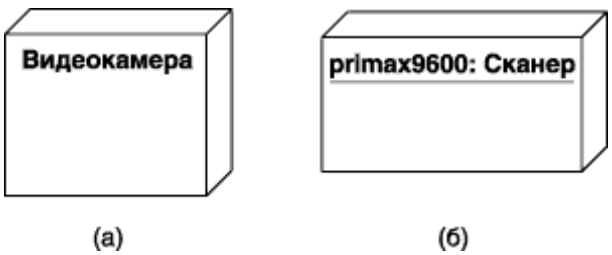


Рисунок 8.1 – Графическое изображение узла на диаграмме развертывания

В первом случае имя узла записывается в форме: <Имя типа узла> без подчеркивания и начинается с заглавной буквы. Во втором – имя узла – экземпляра записывается в виде: <имя узла':' Имя типа узла >, а вся запись

подчеркивается. Имя типа узла указывает на разновидность узлов, присутствующих в модели системы. Так, на представленном рисунке (рис.8.2, а) узел с именем Видеокамера относится к общему типу и никак не конкретизируется. Второй узел (рис.8.2, б) является узлом – экземпляром конкретной модели сканера.

Изображения узлов могут расширяться, чтобы включить дополнительную информацию о спецификации узла. Если дополнительная информация относится к имени узла, то она записывается под этим именем в форме помеченного значения (рис.8.2).

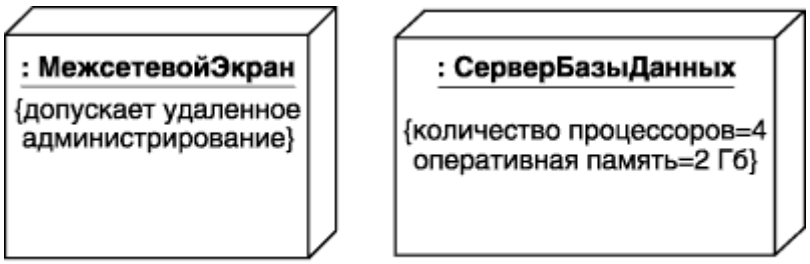


Рисунок 8.2 – Графическое изображение узла-экземпляра с дополнительной информацией в форме помеченного значения

При необходимости явно указать компоненты, которые размещаются или выполняются на отдельном узле, это можно сделать двумя способами. Первый из них позволяет разделить графический символ узла на две секции горизонтальной линией. В верхней секции записывают имя узла, а в нижней - размещенные на этом узле компоненты (рис.8.3, а). Второй способ разрешает показывать на диаграмме развертывания узлы с вложенными изображениями компонентов (рис.8.3, б) Важно помнить, что в качестве таких вложенных компонентов могут выступать только исполняемые компоненты и динамические библиотеки.

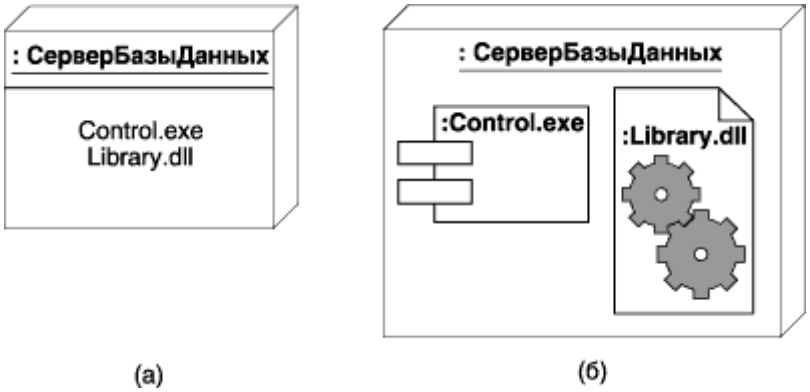


Рисунок 8.3 – Варианты графического изображения узлов-экземпляров с размещаемыми на них компонентами

В качестве дополнения к имени узла могут использоваться различные текстовые стереотипы, которые явно специфицируют назначение этого узла. Для этой цели были предложены следующие текстовые стереотипы:

"processor" (процессор), "sensor" (датчик), "modem" (модем), "net" (сеть), "printer" (принтер) и другие, смысл которых понятен из контекста.

«На диаграммах развертывания допускаются специальные условные обозначения для различных физических устройств, графическое изображение которых проясняет назначение или выполняемые устройством функции. Однако пользоваться этой возможностью следует осторожно, памятуя о том, что основное достоинство языка UML следует из его названия - унификация графических элементов визуализации моделей.

Возможность включения персонала в понятие узла не рассматривается в нотации языка UML, тем не менее, подобное расширение понятие узла позволяет создавать средствами языка UML модели самых различных систем, включая бизнес-процессы и технические комплексы. Действительно, для реализации бизнес-процессов компаний удобно рассматривать в качестве узлов -ресурсов системы организационные подразделения, состоящие из персонала»[12, с.278].

Автоматизация управления техническими комплексами может потребовать рассмотрения в качестве самостоятельного элемента человека-оператора, способного принимать решения в нештатных ситуациях и нести ответственность за возможные последствия этих решений.

Наиболее известны два специальных графических стереотипа для обозначения разновидностей узлов. Первый обозначает ресурсоемкий узел (processor), под которым понимается узел с процессором и памятью, необходимыми для выполнения исполняемых компонентов. Он изображается в форме куба с боковыми гранями, окрашенными в серый цвет (рис.8.4, а).). Второй стереотип в форме обычного куба обозначает устройство (device), под которым понимается узел без процессора и памяти (рис.8.3, б). На этом типе узлов не могут размещаться исполняемые компоненты программной системы.



Рисунок 8.4 – Варианты изображения графических стереотипов узлов

Следует заметить, что кроме графического изображения ресурсоемких узлов и устройств соответствующие узлы можно изображать с помощью обычного символа узла (рис.8.5, а) и дополнительного стереотипа "processor" или "device".

Кроме известных текстовых и графических стереотипов для узлов диаграммы развертывания разработчики могут предложить дополнительные графические стереотипы, которые улучшают наглядность представления диаграмм развертывания. Например, рабочую станцию можно изобразить в

виде ресурсоемкого узла, или в форме рисунка внешнего вида компьютера (рис.8.5, в). Соответственно, сканер также может быть изображен в виде рисунка или фотографии данного устройства.

Соединения и зависимости на диаграмме развертывания

На диаграмме развертывания кроме изображения узлов указываются отношения между ними. В качестве отношений выступают физические соединения между узлами, а также зависимости между узлами и компонентами, которые допускается изображать на диаграммах развертывания.

Соединения являются разновидностью ассоциации и изображаются отрезками линий без стрелок. Наличие такой линии указывает на необходимость организации физического канала для обмена информацией между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, стереотипом, помеченным значением или ограничением. Так, на представленном ниже фрагменте диаграммы развертывания (рис. 8.6) явно определены рекомендации по технологии физической реализации соединений в форме примечания[24].



Рисунок 8.6 – Фрагмент диаграммы развертывания с соединениями между узлами

Кроме соединений на диаграмме развертывания могут присутствовать отношения зависимости между узлом и размещаемыми на нем компонентами. Подобный способ представляет собой альтернативу вложенному изображению компонентов внутри символа узла, что не всегда удобно, поскольку делает этот символ излишне объемным. При большом количестве развернутых на узле компонентов соответствующую информацию можно представить в форме отношения зависимости (рис. 8.7).

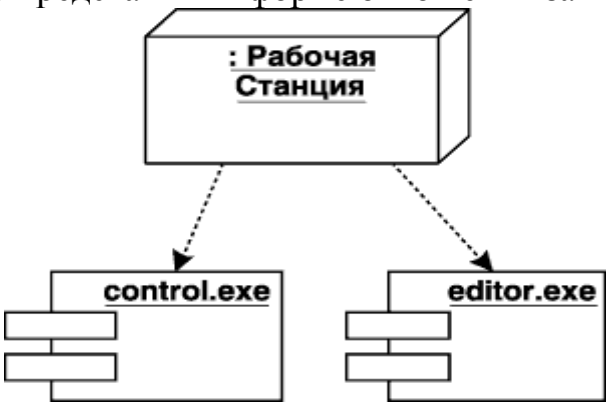


Рисунок 8.7-. Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами

§ 8.2 Рекомендации по построению диаграммы развертывания

Разработка диаграммы начинается с определения расположения всего оборудования, механических и других типов устройств, необходимых для выполнения всех функций системы. Сначала определяются вычислительные узлы системы, имеющие процессор и память. При этом используются существующие стереотипы в языке UML, а при их отсутствии разработчики могут создавать новые стереотипы. Индивидуальные требования к составу оборудования могут быть заданы в виде ограничений и помеченных значений.

Дальнейшие детали диаграммы развертывания связаны с размещением всех исполняемых компонентов плана на узлах системы. Модель должна исключать ситуацию, когда отдельные исполняемые компоненты не размещаются на узлах. Для этого в модель можно добавить дополнительные узлы, содержащие процессор и память. При разработке простых программ, работающих локально на компьютере, диаграмма развертывания не требуется. В более сложных ситуациях диаграмма развертывания строится для следующих приложений.

Во-первых, для моделирования программных систем, реализующих технологию клиент-серверного доступа к данным. Такие системы характеризуются четким разделением полномочий и, следовательно, компонентов между клиентскими рабочими станциями и сервером базы данных. Реализация «тонких» клиентов на простых терминалах или возможность организации доступа к хранилищам данных требует уточнения не только топологии системы, но и состава компонентов.

Во-вторых, для моделирования гетерогенной распределенной архитектуры. Речь идет о корпоративной интрасети, состоящей из сотен компьютеров и других периферийных устройств, работающих на разных платформах и под разными операционными системами. При этом отдельные узлы такой системы могут находиться на расстоянии сотен километров друг от друга, например, центральный офис в столице и филиалы компании в регионах. В этом случае диаграмма развертывания становится важным инструментом для визуализации общей топологии системы и управления миграцией отдельных компонентов между узлами.

В-третьих, моделирование систем реального времени со встроенными микропроцессорами, способными работать автономно. Такие системы могут включать различные дополнительные устройства, обеспечивающие автономность их работы при решении сложных целевых задач. Диаграмма развертывания таких систем позволяет визуализировать состав всех устройств и их соединения в системе[12].

Диаграмма развертывания разрабатывается на заключительном этапе объектно-ориентированного программирования, который характеризует завершение этапа проектирования физического представления. С другой стороны, диаграмма развертывания может быть построена для анализа существующей системы с целью ее дальнейшего уточнения и модификации.

При этом разработка диаграммы на начальных этапах анализа характеризует ее общее направление от физического представления к логическому.

При моделировании бизнес-процессов диаграмма развертывания, помимо компьютеров корпоративной сети, может включать в себя в качестве узлов различное офисное оборудование, такое как факсимильные аппараты, многоканальные телефонные станции, мультиплексоры, экраны презентаций и другие. Более того, каждое устройство может работать как независимо, так и в составе корпоративной сети.

Если в модель необходимо включить интернет-ресурсы, то интернет часто отображается на диаграмме развертывания в виде «облака» с соответствующим именем. Строго говоря, это обозначение не определено в языке UML, но общепринято при разработке моделей веб-приложений.

Язык UML определяет графическую нотацию для каждого элемента канонических диаграмм, но методы графического представления отдельных инструментов имеют свои особенности. Использование конкретного инструментального случая накладывает определенные ограничения на визуализацию моделей программных систем. Некоторые элементы языка UML могут вообще отсутствовать в некоторых Case-инструментах. Выход из этой ситуации может быть связан либо с выбором другого инструмента, поддерживающего последние версии языка UML, либо при упрощении модели на основе ее типизации.

§ 9. Рекомендации по графическому изображению диаграмм языка UML

При графическом представлении диаграмм следует придерживаться следующих основных принципов:

Каждая диаграмма должна представлять собой полное описание соответствующей части моделируемой предметной области. При разработке диаграммы необходимо учитывать все сущности, важные с точки зрения контекста модели и данной диаграммы. Отсутствие некоторых элементов на диаграмме свидетельствует о неполноте модели и может потребовать дальнейшего улучшения.

Все сущности на диаграмме модели должны быть на одном уровне представления. Здесь имеется в виду согласованность не только имен одинаковых элементов, но и возможность вложения отдельных диаграмм друг в друга для достижения полноты представлений. В случае достаточно сложных моделей систем желательно придерживаться стратегии последовательного уточнения или детализации отдельных диаграмм.

Вся информация о сущностях должна быть четко представлена на диаграммах. В языке UML, если на диаграмме отсутствуют некоторые символы, можно использовать их значения по умолчанию.

Не следует включать противоречивые данные. Несовместимость модели может привести к серьезным проблемам при ее реализации и

практическом использовании. Например, при описании отношений агрегации или композиции наличие замкнутых путей вызывает ошибки в программном коде, который будут выполнять соответствующие классы. Наличие элементов с одинаковыми именами и различными атрибутами свойств в одном и том же пространстве имен также приводит к неоднозначной интерпретации и может быть источником проблемы.

«Каждая диаграмма должна быть самодостаточной, чтобы можно было правильно интерпретировать все её элементы и понимать семантику всех используемых графических символов. Любые пояснительные тексты (например, комментарии), не являющиеся неотъемлемыми элементами диаграммы, не должны учитываться разработчиками. В то же время общие элементы диаграммы могут быть уточнены или детализированы в других диаграммах того же типа, образуя вложенные диаграммы или поддиаграммы. Таким образом, модель системы UML представляет собой набор иерархически вложенных диаграмм, детализация которых должна быть достаточной для последующего создания программного кода, реализующего соответствующую системную архитектуру» [12,с.10].

Количество типов диаграмм для конкретной модели приложения не является строго фиксированным. Для простых приложений нет необходимости создавать все типы диаграмм без исключения. Некоторые из них могут отсутствовать в системном проекте, и разработчик не сочтет это ошибкой. Например, системная модель может не содержать диаграммы развертывания для приложения, работающего локально на компьютере пользователя. Важно понимать, что список планов зависит от специфики конкретного системного проекта.

Любая системная модель должна содержать только элементы, определенные в нотации UML. Практика показывает, что этих конструкций достаточно для представления наиболее типичных проектов программных систем. И только при отсутствии необходимых базовых элементов языка UML следует использовать механизмы их расширения для адекватного представления конкретной системной модели. Не допускается переопределение семантики элементов, связанных с базовой нотацией метамодели UML[12].

Список использованных источников

1. Абрамова О. Ф., Д. Н. Лясин Введение в программную инженерию: методические указания к лабораторной работе на тему «Основные сведения о UML и BOUML. Диаграммы вариантов использования [Электронный ресурс]: методические указания / О. Ф. Абрамова// Систем. требования:Windows 95 и выше; ПК с процессором 486+; CD-ROM.
2. Арлоу Д., Нейштадт И. UML 2 и Унифицированный процесс. Практический объектноориентирован ный анализ и проектирование, 2е издание. – Пер. с англ. – СПб: Символ Плюс, 2007. – 624 с., ил
3. Болотовский, Ю. И. ORCAD 9.x, ORCAD 10.x. Практика моделирования / Ю. И. Болотовский, Г. И. Таназлы. – Москва: СОЛОН-ПРЕСС, 2010. – 208 с. – ISBN 978-5-91359-015-2. – Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. – URL: http://www.iprbookshop.ru/65133.html
4. Буч Г., Максимчук Р. А. и др. Объектно-ориентированный анализ и проектирование с примерами приложений/ Пер. с англ. – 3-е изд. – М., СПб., Киев: Вильямс, 2008, 720 с.
5. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя. 2-е изд. М.: ДМК Пресс, 2006, 496 с.
6. Буч, Г. Язык UML. Руководство пользователя. 2-е изд. / Г. Буч, Д. Рамбо, И. Якобсон; пер. с англ. Н. Мухин – М.: ДМК Пресс, 2013. - 496с.: ил.
7. Довбуш Г. Ф.Технология разработки программного обеспечения/ Конспект лекций для студентов четвёртого курса специальности 230105.65. – Программное обеспечение вычислительной техники и автоматизированных систем. – Санкт-Петербург, 2010, с. 131
8. Забелин, Л. Ю. Основы компьютерной графики и технологии трехмерного моделирования: учебное пособие / Л. Ю. Забелин, О. Л. Конюкова, О. В. Диль. – Новосибирск: Сибирский государственный университет телекоммуникаций и информатики, 2015. – 259 с. – Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. – URL: https://www.iprbookshop.ru/54792.html
9. Каюмова А.В. Визуальное моделирование систем в StarUML: Учебное пособие/ А.В. Каюмова. Казань. – Казанский федеральный университет, 2013. – 104с.
10. Иванов Д.Ю., Новиков Ф.А. Унифицированный язык моделирования UML: [учеб. пособие]. СПб.: Изд-во Политехн. ун-та, 2011,229 с.
11. Ларман К. Применение UML 2.0 и шаблонов проектирования. / Пер. с англ. – 3-е изд. – М.: Диалектика, 2019.,736 с.
12. Леоненков А. В. Самоучитель UML. СПб.: БХВ-Петербург, 2004, 427 с.
13. Малиновская, Е. А. Языки программирования. Часть 1: лабораторный практикум / Е. А. Малиновская, Р. А. Рыскаленко. –

Ставрополь: Северо-Кавказский федеральный университет, 2016. – 103 с. – Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. – URL: https://www.iprbookshop.ru/69449.html

14. Малявко, А. А. Формальные языки и компиляторы: учебник / А. А. Малявко. – Новосибирск: Новосибирский государственный технический университет, 2014. – 431 с. – ISBN 978-5-7782-2318-9. – Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. – URL: https://www.iprbookshop.ru/47725.html

15. Мартин Р., Мартин М. Принципы, паттерны и методики гибкой разработки на языке C# : пер. с англ. СПб.: Символ-Плюс, 2011, 768 с.

16. Моисеев А. Н., Литовченко М. И. Основы языка UML: учеб. пособие. – Томск: Издательство Томского государственного университета, 2023, 96 с.

17. НОУ ИНТУИТ | Визуальное моделирование: теория и практика | Информация https://intuit.ru/studies/courses/1041/218/info?ysclid=mk1eec00w9979346449

18. Основы программного моделирования. Практикум: учебное пособие / Ю. В. Бугаев, Л. А. Коробова, С. Н. Черняева, А. С. Чайковский. – Воронеж: Воронежский государственный университет инженерных технологий, 2013. – 68 с. – ISBN 978- 5-00032-016-7. Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. URL: https://www.iprbookshop.ru/47438.html

19. Пальмов, С. В. Методы и средства моделирования программного обеспечения: конспект лекций / С. В. Пальмов. – Самара: Поволжский государственный университет телекоммуникаций и информатики, 2016. – 105 с. – Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. – URL: https://www.iprbookshop.ru/71855.html

20. Пальмов, С. В. Методы и средства моделирования программного обеспечения: методические указания к лабораторным работам / С. В. Пальмов. – Самара: Поволжский государственный университет телекоммуникаций и информатики, 2016. – 33 с. – Текст: электронный // Цифровой образовательный ресурс IPR SMART: [сайт]. – URL: https://www.iprbookshop.ru/71854.html

21. Рамбо Дж., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка. СПб.: Питер, 2007, 544 с.

22. Скотт К. UML. Основные концепции: Быстрое и вместе с тем исчерпывающее введ. в UML. Лег. и простое объяснение ключевых концепций. Большое количество диагр. UML с ясными пояснениями / Кендалл Скотт; [Пер. с англ. О.А. Лещинского]. – М. [и др.]: Вильямс, 2002 (ГПП Печ. Двор). – 138 с.: ил.; 21 см.

23. Фаулер М. UML основы: крат. рук. по стандарт. языку объект. моделирования / Мартин Фаулер; [предисл. Криса Кобрина и др.; [пер. с англ. А. Петухова]. – 3-е изд. – СПб.: Символ, 2005 (ППП Тип. Наука). – 184 с. : ил., табл.; 24 см.

24. Солдатенко М.Н. CASE-технологии информационных систем в

среде StarUM L / Учебно-методическое пособие/- Белгород, 2017 г, 109 с.

25. Столяров А. Проектирование Sequence-диаграмм: руководство для системных аналитиков https://habr.com/ru/articles/924396/

26. Учебник по UML Coderlessons.com. URL: https://coderlessons.com/tutorials/kompiuternoe-programmirovanie/uchebnik-uml/uchebnik-uml

27. Фаулер М. UML. Основы. 3-е изд. СПб.: Символ-Плюс, 2004, 192 с.

28. Хафизов Р. Проектирование измерительно-вычислительных систем медицинского назначения: Конспект лекций. – Йошкар-Ола: Марийский государственный технический университет, 2008. –; 20

29. Хританков А. С., Полежаев В. А., Андрианов А. И. Проектирование на UML. Сборник задач по проектированию программных систем. 2-е изд. Екатеринбург: Издательские решения, 2017, 240 с.

30. Якобсон А., Буч Г., Рамбо Д. Унифицированный процесс разработки программного обеспечения. / Пер. с англ. – СПб.: Питер, 2002, 496 с.

31. Collaboration Diagrams | Unified Modeling Language(UML) – GeeksforGeeks https://www.geeksforgeeks.org/system-design/collaboration-diagrams-unified-modeling-languageuml/?ysclid=mk1e8wa9ui496508176

32. Welcome to UML Web Site! URL: http://www.uml.org

33. https://staruml.io/

ГЛОССАРИЙ

1. Action (активность) – выполняемые задачи или выполнение определенных действий в течение жизни объекта и обозначается прямоугольником со скругленными углами.

2. Alt (условное действие) – указывает на логику ветвления в зависимости от условия. Состоит из двух или более частей, одна из которых активируется в зависимости от условий

3. Case – инструменты – естественная среда для применения и визуализации и методологии ООАП

4. Loop (цикл) – указывает на блок повторяющихся действий. Включает состояние выхода из цикла.

5. Neg – альтернативный сценарий, который не следует реализовывать

6. Opt (опциональное действие) – указывает на блок, который может быть выполнен или пропущен в зависимости от условия.

7. Par (параллельная операция) – указывает на параллельную работу нескольких блоков.

8. Ref – ссылается на другую диаграмму последовательности

9. Swimlanes (дорожки) – это визуальный элемент на диаграмме активности, который направлен на группировку действий по ролям, ресурсам, элементам организации или местоположению

10. UML (Unified Modeling Language) – это язык для проектирования и документирования программных компонентов, бизнес-процессов и других систем

11. Абстракция – это анализ объекта с точки зрения его наиболее важных характеристик, которые отличают его от всех других типов объектов и от других объектов

12. Абстракция (abstraction) – характеристика сущности, которая отличает ее от других сущностей. Абстракция определяет границу представления соответствующего элемента модели и применяется для определения фундаментальных понятий ООП, таких как класс и объект

13. Автомат (Statemachine) – это описание последовательности состояний, через которые проходит объект на протяжении всего жизненного цикла, реагируя на события, – в том числе описание реакций на эти события.

14. Агрегация – это отношения между целым и частью, при этом целое и часть могут быть разделены, т.е. объекты-члены являются частью общего объекта, но объект-член может существовать независимо от общего объекта

15. Акторы – это внешние сущности, взаимодействующие с системой, такие как пользователи или другие системы

16. Ассоциация – связь между актором и вариантом использования, которая описывает роль актора при взаимодействии с конкретным вариантом использования

17. Атрибут – свойство класса, описывающее его характеристики.

18. Бизнес-актор – это отдельный человек, группа, организация, компания или система, взаимодействующие с моделируемой бизнес-

системой, но не являющиеся её частью, то есть не являющиеся частью моделируемой системы.

19. Бизнес-вариант использования – это вариант использования, определяющий последовательность действий моделируемой системы для реализации отдельного бизнес-процесса.

20. Видимость – уровень доступа к атрибутам и методам класса

21. Входное действие (entry action) – поведение, которое выполняется, когда объект переходит в данное состояние

22. Выражение действия (action expression) представляет собой вызов операции или передачу сообщения, имеет атомарный характер и выполняется сразу после срабатывания соответствующего перехода до начала действий в целевом состоянии.

23. Границы – это точки, определяющие границы взаимодействия системы с внешним миром, такие как пользовательский интерфейс или API.

24. Деятельность (activity) – поведение, реализуемое объектом, пока он находится в данном состоянии, прерываемое поведение.

25. Диаграмма – это графическое представление набора элементов модели в виде последовательного графа с вершинами и ребрами (дугами), которому присваивается определенная семантика

26. Диаграмма деятельности – это графическое представление рабочих процессов поэтапных действий и действий с поддержкой выбора, итерации и параллелизма.

27. Диаграмма кооперации (кооперативная диаграмма) – вид диаграммы взаимодействия, ориентированный на визуализацию взаимодействия и отношений между объектами в системе. Главная задача – показать, как объекты обмениваются сообщениями для достижения нужной цели или реализации определенных вариантов использования.

28. Диаграмма последовательности (Sequence Diagram) – отражает последовательность взаимодействия между объектами, показывает поток событий, происходящих в вариантах использования.

29. Диаграмма развертывания (deployment diagram) – диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов, предназначена для визуализации элементов и компонентов программы, существующих только на этапе ее исполнения (run-time).

30. Диаграмма состояний – описывает поведение объекта через состояния и переходы, подходят для отображения жизненного цикла объекта, но они не показывают взаимодействия между объектами.

31. Диаграммы активности (**Activity diagrams**) подходят для описания общего потока действий, особенно если уделяется внимание действиям и логике принятия решений, но они плохо передают взаимодействие между участниками.

32. Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов.

33. Сообщение (message) – это средство, с помощью которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций

34. Зависимость отражает связь между объектами по применению, когда изменение поведения одного класса может повлиять на поведение другого. В большинстве случаев зависимости отражаются в методах класса, использующих в качестве параметра объект другого класса

35. Императивное (imperative) сообщение – это сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

36. Индивидуальность означает, что каждый объект уникален, даже если его состояние идентично состоянию другого объекта

37. Инкапсуляция – принцип сокрытия внутренней структуры объекта, характеризует сокрытие отдельных деталей внутреннего устройства классов от внешних по отношению к нему объектов или пользователей.

38. **Инструменты для создания UML-диаграмм** – программы, которые поддерживают создание, редактирование и управление диаграммами.

39. Интерфейс (включая абстрактный класс) – это набор методов. В отношениях реализации класс реализует интерфейс, а методы в классе реализуют все методы, объявленные в интерфейсе

40. Информационное (informative) сообщение – это сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

41. Класс – это описание набора объектов с общими атрибутами, операциями и семантикой. Графически класс представляется в виде прямоугольника, разделённого на три части, где записываются его имя, атрибуты и операции.

42. Класс-сущность содержит информацию, хранимую постоянно. Используется для моделирования данных и поведения с длинным жизненным циклом.

43. Композиция рассматривается как отношения между целым и частью, но целое и часть не могут существовать без друг друга. Это частный случай агрегации, при котором составные части целого имеют такое же время жизни, что и само целое, и уничтожаются вместе с уничтожением целого

44. Компонент – это физически существующая часть системы, которая обеспечивает реализацию классов и отношений моделируемой программной системы, а также её функциональное поведение

45. Конечное состояние (finalstate) – разновидность псевдосостояния, обозначающее прекращение процесса изменения состояний конечного автомата

46. Контроллеры – контролируют поток данных и логику системы, обрабатывают запросы от участников и перенаправляют их соответствующим сущностям.

47. Метод – операция, которую может выполнить объект.

48. Модель – это абстрактное представление физической системы на оправленном языке с помощью специальных графических средств, рассматриваемая с определенной точки зрения.

49. Наследование (обобщение) используется для описания взаимосвязи между родительскими и дочерними классами. Родительский класс называется базовым классом, а подкласс – производным классом.

50. Начальное состояние (startstate) – разновидность псевдосостояния, обозначающее начало выполнения процесса изменения состояний конечного автомата или нахождения моделируемого объекта в составном состоянии.

51. **Нотация UML** – стандартных символов и правил для UML-диаграмм. Условно элементы можно поделить на три группы: графические (прямоугольники, овалы), связи (разные типы линий, которые показывают, как графические элементы относятся друг к другу) и специальные обозначения (технические метки для уточнения деталей).

52. Обобщение – отношение между классами, где один класс является подтипом другого.

53. Объединение (Merge Node) – это слияние нескольких веток в одну, позволяет вернуться к общему сценарию после выполнения альтернативных путей.

54. Объект – это некоторая сущность реального мира или концептуальная (абстрактная) сущность

55. Объект – экземпляр класса с определёнными атрибутами и поведением.

56. Объектно-ориентированное программирование (ООП) – это набор принципов, технологий и инструментов для создания программных систем, основанных на архитектуре взаимодействия объектов

57. Объектно-ориентированный анализ и проектирование (ООАП) – это технология разработки программных систем, основанная на объектно-ориентированной методологии представления предметной области в виде объектов как экземпляров соответствующих классов.

58. Ограничение (constraint) – это некоторые логические условия, которые ограничивают семантику выбранного элемента модели.

59. Отношение – связь между элементами модели

60. Отношение зависимости между двумя вариантами использования, при котором один вариант (базовый) обязательно включает в себя функциональность другого варианта как составной части называется отношением включения

61. Отношение расширения определяет взаимосвязь базового варианта использования с другим вариантом, который реализуется не всегда, а только при выполнении определённых условий.

62. Пакет (package) – общецелевой механизм для организации различных элементов модели в группы. Пакет в логическом представлении модели - это объединение классов или других пакетов

63. Параллельное выполнение (Fork Node) позволяет определить независимо выполняемые действия

64. Поведение – это характеристика класса/объекта, которая заключается в том, как объект ведет себя и реагирует (то есть, как он изменяет свое состояние). Поведение выражается через состояние объекта и передачу сообщений

65. Подпакет (subpackage) – пакет, который является составной частью другого пакета.

66. Полиморфизм как один из принципов методологии объектно-ориентированного программирования предполагает, что, действия, которые выполняются одними и теми же методами, зависят от того к какому классу относится тот или иной метод и, как следствие, могут различаться наследования рассматривается как возможность создавать иерархии классов: классы потомки могут наследовать данные и методы родительских классов.

67. Помеченное значение (tagged value) – конкретное определение свойства в виде пары «имя-значение». При указании значения само имя называется тегом.

68. Поток управления/переход (Control Flow) – указывает переход из одного состояния в другое или по завершении выполнения определенного действия в начало другого и обозначается стрелкой.

69. Прецендент – описание взаимодействия между актором и системой

70. Псевдосостояние (pseudo-state) – вершина в конечном автомате, которая имеет форму состояния, но не обладает поведением

71. Решение/Ветвление (Decision Node) позволяет показать зависимость дальнейшей работы от внешних условий или решений

72. Синхронизация (Join Node) – слияние нескольких параллельных процессов в один поток, обозначается линией, в которую входят несколько стрелок.

73. Сообщение-запрос (interrogative) – это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

74. Состояние – определенная конфигурация объекта в конкретный момент времени

75. Состояние (State) – это ситуация в жизни объекта, на протяжении которой он удовлетворяет некоторому условию, осуществляет определенную деятельность или ожидает какого-то события.

76. Состояние объекта–это одно из условий, в котором он может находиться. Состояние обычно изменяется со временем и характеризуется набором свойств, которые называются атрибутами

77. Сотрудник – это человек, который работает в моделируемой бизнес-системе, взаимодействует с другими работниками и участвует в бизнес-процессе моделируемой системы.

78. Стереотип (stereotype) – новый элемент модели, расширяющий семантику метамодели. Стереотипы должны основываться на существующих типах или классах и описываются в метамодели UML.

79. Сторожевое условие (guard) определяют, когда переход может, а когда не может осуществиться.

80. Сущности – часто представляют данные и состояние системы, такие как базы данных или другие хранилища данных.

81. Сценарий использования (UC) – это подробное описание того, как пользователь взаимодействует с системой для достижения конкретной цели.

82. Узел (node) представляет собой физически существующий элемент системы, который может обладать вычислительным ресурсом или являться техническим устройством.

83. **Уничтожение объекта – э**лемент обозначающий конец его линии жизни в этом взаимодействии, т.е. Момент, когда объект перестает иметь значение в последовательности

84. Язык UML – это общий язык визуального моделирования, предназначенный для спецификации, визуализации, проектирования и документирования программных компонентов, бизнес-процессов и других систем.

БИДЖИЕВА Сапият Ханапиевна

УНИФИЦИРОВАННЫЙ ЯЗЫК МОДЕЛИРОВАНИЯ UML

УЧЕБНОЕ ПОСОБИЕ
для обучающихся 3 курса по направлению подготовки
09.03.04 Программная инженерия

Корректор Чагова О. Х.
Редактор Чагова О. Х.

Сдано в набор 15. 01. 2026 г.
Формат 60х84/16
Бумага офсетная
Печать офсетная
Усл.печ. 6,27
Заказ № 5308
Тираж 100 экз.

Оригинал–макет подготовлен
В библиотечно-издательском центре СКГА
369000, г. Черкесск, ул. Ставропольская, 36