МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра Телевидения и управления (ТУ)

Утверждаю:
Зав. Кафедрой ТУ
Д.т.н., с.н.с.
_____Т.Р. Газизов
«___»_____2016 г.

Современные технологии анализа и проектирования информационных систем

Учебно-методическое пособие по практическим занятиям для студентов направления 43.03.01 «Сервис» профиля «Информационный сервис»

Разработчк:

Доцент каф. ТУ

_____С.П. Куксенко

Томск – 2016

Куксенко С.П.
Современные технологии анализа и проектирования информационных систем: учеб. метод. пособие. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2016. – 101 с.

Представлены методические материалы практикума, посвященного анализу и проектированию информационных систем с помощью структурного и объектно-ориентированного подходов. Рассмотрены принципы работы с CASE-средствами Ramus Educational и StarUML.

Предназначено для бакалавров технических вузов, обучающихся по направлению подготовки 43.03.01 «Сервис» профиля «Информационный сервис» в рамках дисциплины «Современные технологии анализа и проектирования информационных систем».

Оглавление

1. Разработка технического задания

Цель работы – изучить основные принципы и получить базовые навыки подготовки технических заданий на разработку информационной системы (программного продукта).

В России при создании информационных систем (программных продуктов) используется следующие ГОСТ 34.003-90 «Автоматизированные системы. Термины и определения», ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы», ГОСТ 34.601-90 «Автоматизированные системы. Стадии создания».

При разработке информационной системы (прикладного программного продукта) существует два основных момента, которые требуют обязательного документального подтверждения: договорные отношения (контракт) и требования к конечному результату – техническое задание (ТЗ). Основная цель написания ТЗ – устранение двусмысленностей о том, что именно будет являться конечным продуктом. Юридически техническое задание оформляется как приложение к договору оказания услуг по разработке и подписывается обеими сторонами. Техническое задание – исходный документ для разработки информационной системы (программного продукта), содержащий основные технические требования, предъявляемые к продукту и исходные данные для разработки. В ТЗ указываются назначение продукта, область его применения, целевая аудитория, стадии разработки проектной и программной документации, её состав, сроки исполнения и т.д., а также особые требования, обусловленные спецификой данного продукта либо условиями его эксплуатации. Как правило, ТЗ составляют на основе анализа результатов предварительных исследований, расчётов и моделирования. Вызвано это тем, что крупные проекты требуют серьезного проектного исследования. Обычно на эти исследования выделяется отдельный бюджет и порой не меньший, чем на непосредственно разработку проекта. Связано это с тем, что точную оценку стоимости крупного проекта можно дать только после точного его описания (которое и составляет ТЗ), а заказчик может отказаться от дальнейшего сотрудничества, хотя разработчик уже понес существенные трудозатраты. Не всякий заказчик готов к такой постановке вопроса. Как правило, Заказчик не является профессионалом в области высоких технологий, и задача им ставится на общем уровне: «мы бы хотели увидеть вот это, может это, а может еще и это». При этом зачастую представители заказчика вообще не придают особого значения составлению ТЗ на разработку проекта.

Часто приходится проектные исследования сводить к минимуму или часть исследований проводить бесплатно в надежде на получение крупного проекта, что в конечном итоге может негативно сказаться на эффективности работ. Конечно, любому заказчику проще не составлять ТЗ, а вносить коррективы по ходу работ, однако такой подход в корне не устраивает любого разработчика, поскольку не позволит оценить затраты на оказание услуг и приведет к убыткам.

Заказчик смотрит на проект с точки зрения выгод для бизнеса. Разработчик – с точки зрения технических проблем и объема работ. В итоге Заказчик стремится к тому, чтобы все получилось максимально хорошо и красиво, а Разработчик – чтобы проект потребовал от него минимум усилий. Если что-то не было заранее оговорено и где-то записано – исполнитель наверняка этого не сделает. В итоге для того, чтобы все получилось так, как нужно заказчику, ему приходится раз за разом «выбивать» из разработчика то, что, как

ему казалось, изначально планировалось сделать. Так как ТЗ вносит ясность, что и в какие сроки будет реализовано, его разработка и подписание несет выгоду для обеих сторон.

1 Получение заказчиком и исполнителем ясного представления о готовом продукте.
В ТЗ максимально точно и подробно описывается процесс работы над проектом и конечные результаты, которые должны быть получены на каждом этапе. В нем, вплоть до мельчайших деталей, прописывается все то, что хочет получить заказчик. Причем представители заказчика должны принимать непосредственное участие в разработке, в противном случае исполнитель напишет то, что выгоднее ему и сократит свои издержки за счет возможностей конечного продукта. При этом лучше избегать варианта, когда ТЗ полностью разрабатывается заказчиком, так как он наверняка упустит некоторые моменты, важные для технической реализации проекта. Кроме того, хорошо разбираясь в своей предметной области, заказчик может считать некоторые вещи само собой разумеющимися, в то время как для исполнителя они будут неочевидными. ТЗ – это плод совместной работы, компромисс между желаниями заказчика и возможностями исполнителя.

2 Заказчик может оценить, сколько на проект потребуется времени.
Если у нет четкой схемы проекта - нельзя даже приблизительно сказать, сколько времени он потребует. Заказчик может думать, что проект займет максимум месяц, а выйдет так, что он затянется на год. Наоборот обычно не бывает – сроки разработки технологических продуктов сложно переоценить. Это особенно критично, когда на сроки реализации завязываются другие бизнес-процессы. ТЗ позволяет достаточно точно оценить, сколько времени займет разработка проекта. Имея на руках техническое задание, исполнитель может оценить собственные возможности и оповестить заказчика, сколько ему потребуется времени на реализацию. Мало того, если исполнитель не в силах реализовать необходимый функционал, он сможет сказать об этом до того, как заказчик заплатит ему деньги.

3 Заказчик может оценить, сколько на проект потребуется денег.
Грамотно составленное ТЗ позволяет достаточно точно определить возможные дополнительные издержки. Зная сколько времени потребуется для работы над проектом, можно определить стоимость труда исполнителей. Не имея точных данных о продолжительности проекта невозможно определить срок, на который необходимо нанять, например, высококлассного программиста или специалиста по базам данных. Зная технические требования проекта, можно сделать выводы, какие сервера, программное обеспечение и лицензии потребуются. Это очень важный момент. Если не озаботиться этим вопросом заранее, запросто может сложиться ситуация, при которой у заказчика на руках будет готовый проект, но необходимые лицензии стоят в десятки раз больше, чем вся разработка проекта. То же самое касается техники – достаточно простой в реализации проект может потребовать дорогостоящего оборудования, каналов связи, обслуживающего персонала.

4 Возможность организации процесса приема-передачи готового продукта. Заказчик получает возможность требовать от исполнителя соответствия продукта всем условиям, оговорённым в ТЗ.

Процесс передачи продукта от заказчика к исполнителю может быть организован в виде приемочного тестирования – последовательной проверки готового продукта на предмет соответствия требованиям ТЗ.

5 Исполнитель может глубже понять суть задачи, показать заказчику «технический облик» будущего программного продукта.

6 Снижение числа проблем между заказчиком и исполнителем, связанных с изменением требований в результате их неполноты или ошибочности (на всех стадиях и этапах создания, за исключением испытаний).

7 Исполнитель может спланировать выполнение проекта и работать по намеченному плану.

8 Исполнитель получает возможность отказаться от выполнения работ, не указанных в ТЗ, в рамках уже заключенного контракта и требовать заключение дополнительного контракта и нового ТЗ.

9 Заказчик может не заниматься контролем исполнителя по ходу работ в режиме реального времени.

Чем подробнее ТЗ, тем меньше разночтений возникнет между заказчиком и исполнителем. Имея точное описание того что необходимо создать, исполнитель сможет работать над проектом не отрывая заказчика от дел по мелочам. Точно так же заказчику не потребуется постоянно руководить работой исполнителя и постоянно говорить ему, что нужно делать. Когда весь проект или какая-то его логическая часть завершена, заказчику достаточно сверить воплощение с ТЗ и своим видением проекта. То есть, весь процесс контроля разбивается на крупные этапы, рутина сводится к минимуму, а эффективность работы стремится к максимуму.

10 Заказчик меньше зависит от конкретного исполнителя.

Имея техническое описание того, как должен работать проект, заказчик перестает зависеть от непосредственного исполнителя. Заказчик всегда может передать разработку или ее часть другому исполнителю и при этом не бояться того, что возникнут разночтения – все требования к проекту жестко зафиксированы в ТЗ. Помимо этого заказчик может оценить эффективность выбора исполнителя, технологической платформы, общего подхода к реализации проекта. Он может обратиться к третьей стороне для того, чтобы провести аудит проекта, получить рекомендации, сделать работу над проектом более эффективной и снизить затраты.

В случае если задание на разработку продукта дается собственным программистам, находящимся в штате компании и получающим фиксированный оклад, ТЗ оказывается весьма полезным, поскольку позволяет существенно сократить время разработки. Если время на исправление ошибок, проводимое на этапе согласования ТЗ, принять за единицу, то внесение изменений в уже законченный проект обычно оценивают десятками. Кроме того, требования проекта определяют людей, которые требуются для его реализации. Расписав проект по пунктам, можно сделать вывод о необходимости тех или иных специалистов – дизайнеров, программистов, системных администраторов, аналитиков.

Объем ТЗ зависит от сложности разрабатываемого продукта и может колебаться от одной до сотни страниц. Согласно ГОСТ 34.602-89 рекомендуется следующая структура ТЗ:

- общие сведения;
- назначение и цели создания (развития) системы;

- характеристика объектов автоматизации;
- требования к системе;
- состав и содержание работ по созданию системы;
- порядок контроля и приемки системы;
- требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие;
- требования к документированию;
- источники разработки.

В табл. приведены типовые требования к составу и содержанию ТЗ.

№ п\п.	Раздел	Содержание
1	Общие сведения	• полное наименование системы и ее условное обозначение шифр темы или шифр (номер) договора; • наименование предприятий разработчика и заказчика системы, их реквизиты • перечень документов, на основании которых создается ИС • плановые сроки начала и окончания работ • сведения об источниках и порядке финансирования работ • порядок оформления и предъявления заказчику результатов работ по созданию системы, ее частей и отдельных средств
2	Назначение и цели создания (развития) системы	• вид автоматизируемой деятельности • перечень объектов, на которых предполагается использование системы • наименования и требуемые значения технических, технологических, производственно-экономических и др. показателей объекта, которые должны быть достигнуты при внедрении ИС
3	Характеристика объектов автоматизации	• краткие сведения об объекте автоматизации • сведения об условиях эксплуатации и характеристиках окружающей среды
4	Требования к системе	Требования к системе в целом: • требования к структуре и функционированию системы (перечень подсистем, уровни иерархии, степень централизации, способы информационного обмена, режимы функционирования, взаимодействие со смежными системами, перспективы развития системы) • требования к персоналу (численность пользователей, квалификация, режим работы, порядок подготовки) • показатели назначения (степень приспособляемости системы к изменениям процессов управления и значений параметров) • требования к надежности, безопасности, эргономике, транспортабельности, эксплуатации, техническому обслуживанию и ремонту, защите и сохранности информации, защите от внешних воздействий, к патентной чистоте, по стандартизации и унификации Требования к функциям (по подсистемам): • перечень подлежащих автоматизации задач • временной регламент реализации каждой функции

		• требования к качеству реализации каждой функции, к форме представления выходной информации, характеристики точности, достоверности выдачи результатов • перечень и критерии отказов Требования к видам обеспечения: • математическому (состав и область применения мат. моделей и методов, типовых и разрабатываемых алгоритмов) • информационному (состав, структура и организация данных, обмен данными между компонентами системы, информационная совместимость со смежными системами, используемые классификаторы, СУБД, контроль данных и ведение информационных массивов, процедуры придания юридической силы выходным документам) • лингвистическому (языки программирования, языки взаимодействия пользователей с системой, системы кодирования, языки ввода-вывода) • программному (независимость программных средств от платформы, качество программных средств и способы его контроля, использование фондов алгоритмов и программ) • техническому • метрологическому • организационному (структура и функции эксплуатирующих подразделений, защита от ошибочных действий персонала) • методическому (состав нормативно-технической документации)
5	Состав и содержание работ по созданию системы	• перечень стадий и этапов работ • сроки исполнения • состав организаций – исполнителей работ • вид и порядок экспертизы технической документации • программа обеспечения надежности • программа метрологического контроля
6	Порядок контроля и приемки системы	• виды, состав, объем и методы испытаний системы • общие требования к приемке работ по стадиям • статус приемной комиссии
7	Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие	• преобразование входной информации к машиночитаемому виду • изменения в объекте автоматизации • сроки и порядок комплектования и обучения персонала
8	Требования к документированию	• перечень подлежащих разработке документов • перечень документов на машинных носителях
9	Источники разработки	документы и информационные материалы, на основании которых разрабатывается ТЗ и система

Как правило, ТЗ составляется компанией – будущим разработчиком – после проведения интервью с заказчиком. В ходе интервью (одной или ряда встреч), представители разработчика выясняют те моменты, которые затем найдут отражение в документе. На интервью со стороны заказчика желательно присутствие всех

заинтересованных в создании и всех, кто будет причастен к его исполнению. Если в компании-заказчике есть грамотные IT-специалисты, то их следует привлечь к согласованию ТЗ. Если же ситуация иная, а речь идет о крупном проекте, бывает целесообразно привлечение независимого эксперта в качестве консультанта, который оценит качество и полноту составления технического задания и защитит интересы заказчика.

Самостоятельная работа.

1 Выбрать предметную область (приложение).
2 Определить информационную систему, автоматизирующую некоторый процесс в выбранной предметной области.
3 Подготовить ТЗ на разработку данной системы (включить раздел календарного плана, содержащий сетевой график и диаграмму Ганта, выполненную с помощью инструментального средства Gantt Project).

2. Методы структурного проектирования: IDEF0 и DFD

2.1 Знакомство с пакетом Ramus Educational

Цель работы – ознакомиться с основами методологии IDEF0 и работы в пакете Ramus Educational.

Методология IDEF0 (Integrated DEFinition) представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели предметной области. Данная методология отображает функциональную структуру системы, т.е. производимые ею действия и связи между этими действиями.

Данная методология применяется при создании новых систем для определения требований и функций и затем для разработки системы, удовлетворяющей требованиям и реализующей функции. Для действующих систем эта методология может использоваться для анализа функций, выполняемых системой, а также для наглядного представления «механизмов», посредством которых эти функции осуществляются. Основной сферой применения методологии IDEF0 является предпроектное обследование и анализ системы.

Методология IDEF0 основана на методе SADT Дугласа Т. Росса. В рамках проекта ICAM, организованного военными ведомствами США с целью разработки подходов, обеспечивающих повышение эффективности проектирования благодаря систематическому внедрению компьютерных технологий, метод SADT и некоторые аспекты его применения были стандартизированы, после чего получили название методологии IDEF0. В соответствии с проектом ICAM было разработано семейство методологий IDEF, в которое кроме методологии создания функциональной модели сложной системы IDEF0 вошли еще две самостоятельных методологии моделирования:

- IDEF1 – методология создания информационной модели производственной среды или системы (основана на реляционной теории Кодда и использование ER – диаграмм Чена);
- IDEF2 – методология создания динамической модели производственной среды или системы.

Модель IDEF0 представляет собой набор диаграмм с поддерживающей их документацией, включающей сопровождающие тексты и словарь. Диаграммы модели декомпозируют сложную систему на составные части. Первоначальная (исходная, корневая) диаграмма является наиболее общим и наиболее абстрактным описанием всей системы в целом. Она показывает основную функциональную составляющую системы в виде блока. Детали (компоненты) каждого из основных блоков показаны на других диаграммах в виде блоков. Далее они могут быть превращены в более подробные диаграммы, до тех пор, пока не будет достигнута требуемая степень детализации.

Блоки представляют собой функции (действия, процессы или операции), а входящие и исходящие из них стрелки представляют объекты (информацию, предметы). Блок и его стрелки на диаграмме рассматриваемого уровня описывается более подробно блокам и стрелками диаграмм нижнего уровня.

Блоки-функции показывают, что должно выполняться, причем без идентификации каких-либо других аспектов, например, таких как потребности в них или средства их осуществления. Наименование функций записывается внутри блока. Оно должно содержать существительное, обозначающее действие. Блоки нумеруются в нижнем углу.

Стрелки на диаграмме играют роль интерфейсов (связей) блоков с внешней для них средой. Каждая из стрелок имеет метку, характеризующую её. Назначение стрелок зависит от стороны блока, в которую стрелка входит или выходит (рис. 2.1):

Входящие стрелки слева от блока представляет собой предметы (материальные объекты) или информацию (информационные объекты), необходимые для выполнения функции. Это сырье, материалы, исходные данные или «вход» функции (стрелка типа I – input).

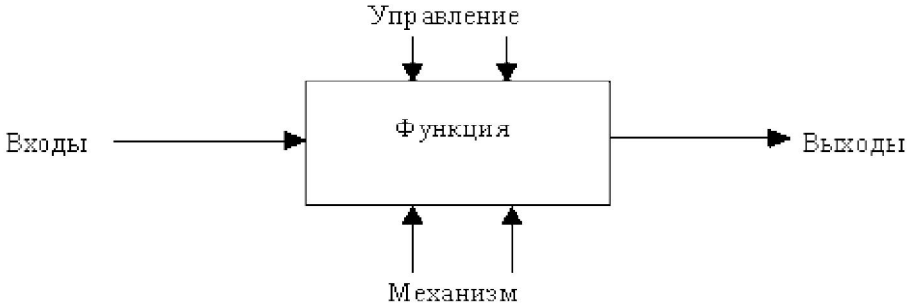


Рисунок 2.1 – Функциональный блок и интерфейсные дуги

Выходящие стрелки справа из блока показывают предметы или данные, полученные в результате выполнения функции блока. Это результат (стрелки типа O – output). Функции преобразуют объекты слева направо (от входа к выходу). Таким образом, блок представляет собой переход от состояния «до» к состоянию «после».

Входящие стрелки сверху блока является управлением, описывающим условия или данные, которые управляют выполнением функции (тип C – control). Как и входы, они тоже могут являться информацией, но назначение входа и управления различны. Их разделение является важным для понимания работы системы.

Входящая снизу стрелка представляет собой «механизм», обозначающая собой либо человека, либо некоторое средство, выполняющее функцию (M – mechanism).

Выход и вход показывают, что и из чего делается функцией, управление показывает, как и почему это делается, а механизм показывает, кем и с помощью чего это делается.

Стрелки на диаграмме IDEF0 означают ограничения, задаваемые связанными с ними объектами (предметами или данными). Они не представляют собой поток или последовательность. Соединяя выход одного блока с входом другого, они показывают ограничения. Блок, получающий объекты, «ограничен» в том смысле, что функция не может быть выполнена, пока не будут получены объекты, производимые другими блоками. Стрелки, входящие в блок, показывают все объекты, которые необходимы для выполнения функции.

Несколько функций на диаграмме могут выполняться одновременно, если удовлетворены все ограничивающие условия. Ни последовательность, ни время не являются точно определенными в IDEF0. Обратная связь, итерация, непрерывные процессы, перекрытие (во времени) функций легко отображаются стрелками.

Стрелки могут разветвляться или соединяться. Каждая из ветвей может представлять один и тот же объект или различные объекты одного и того же типа.

Отсюда следует, что IDEF0-модели – это ни блок-схемы, ни просто диаграммы потоков данных, а предписывающие диаграммы, которые представляют входные/выходные преобразования, а также указывают правила этих преобразований. Примеры типичных наименований блоков и стрелок приведены в следующей таблице.

Функции		
планирование	разработка	классификация
управление	проектирование	измерение
контроль	изготовление	редактирование
расчет	производство	учет
продажа	реализация	сбор
Управление		
инструкции	требования	чертеж
стандарты	запросы	правило
указания	заявки	
задания	план	
Механизм		
персонал	компьютер	информационная система
фирма	станок	программное обеспечение
заказчик	аппаратура	подрядчик
инструмент	оператор	оснастка

Методологии IDEF послужили основой для создания нескольких CASE-продуктов. Один из них Ramus. Пакет Ramus – графическая среда для проектирования и моделирования сложных систем широкого назначения. Ramus Educational – это бесплатный аналог Ramus с некоторыми ограничениями (www.ramussoftware.com), который может быть использован для создания диаграмм в формате IDEF0 и DFD. Формат файлов Ramus Education полностью совместим с форматом файла коммерческой версии Ramus.

Общая схема составления функциональной модели состоит из следующих этапов:

1 построение модели (разработка функциональной диаграммы; заполнение глоссария дополнительными определениями; дополнение диаграммы гипертекстом);

2 проверка синтаксиса модели (проверка на наличие связей, на идентификаторы функций и связей, на управление).

Чтобы начать создание новой IDEF0-модели необходимо:

3 запустить Ramus, при первом запуске программа предлагает создать новый файл или открыть уже существующий (рис. 2.2);

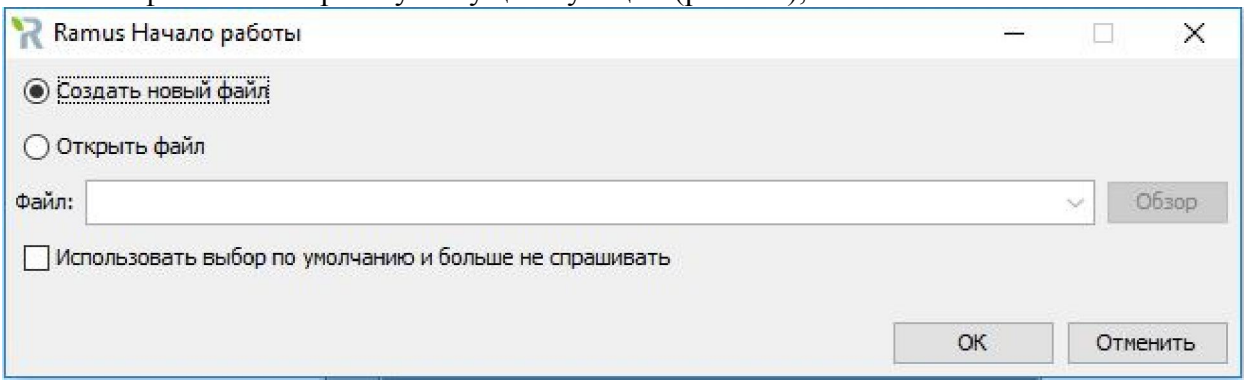


Рисунок 2.2 – Начало работы

4 после того как выбрано «Создать новый файл» в появившемся диалоговом окне «Мастер «Свойства проекта» для создания информационной модели нужно выбрать IDEF0 (рис 2.3);

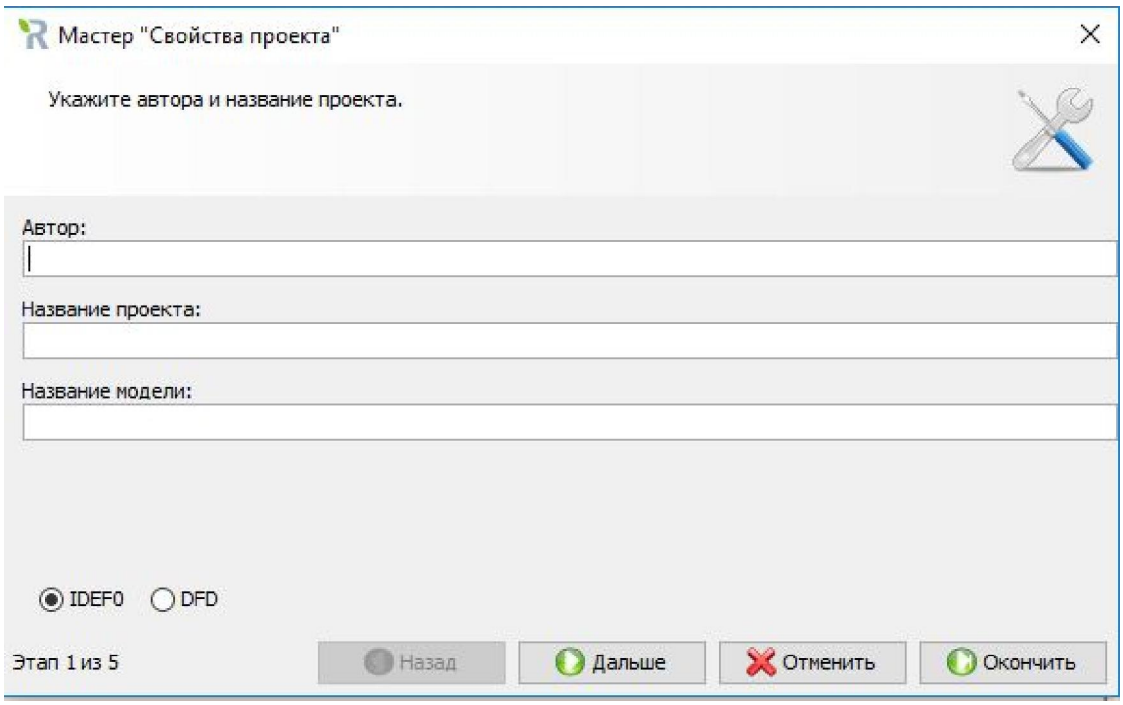


Рисунок 2.3 – Диалоговое окно «Мастер «Свойства проекта»

5 подтвердить выбор клавишей «Окончить»;

6 откроется начальная IDEF0-страница (рис. 1.4) – контекстная диаграмма. В каждой IDEF0-модели можно иметь только одну контекстную диаграмму.

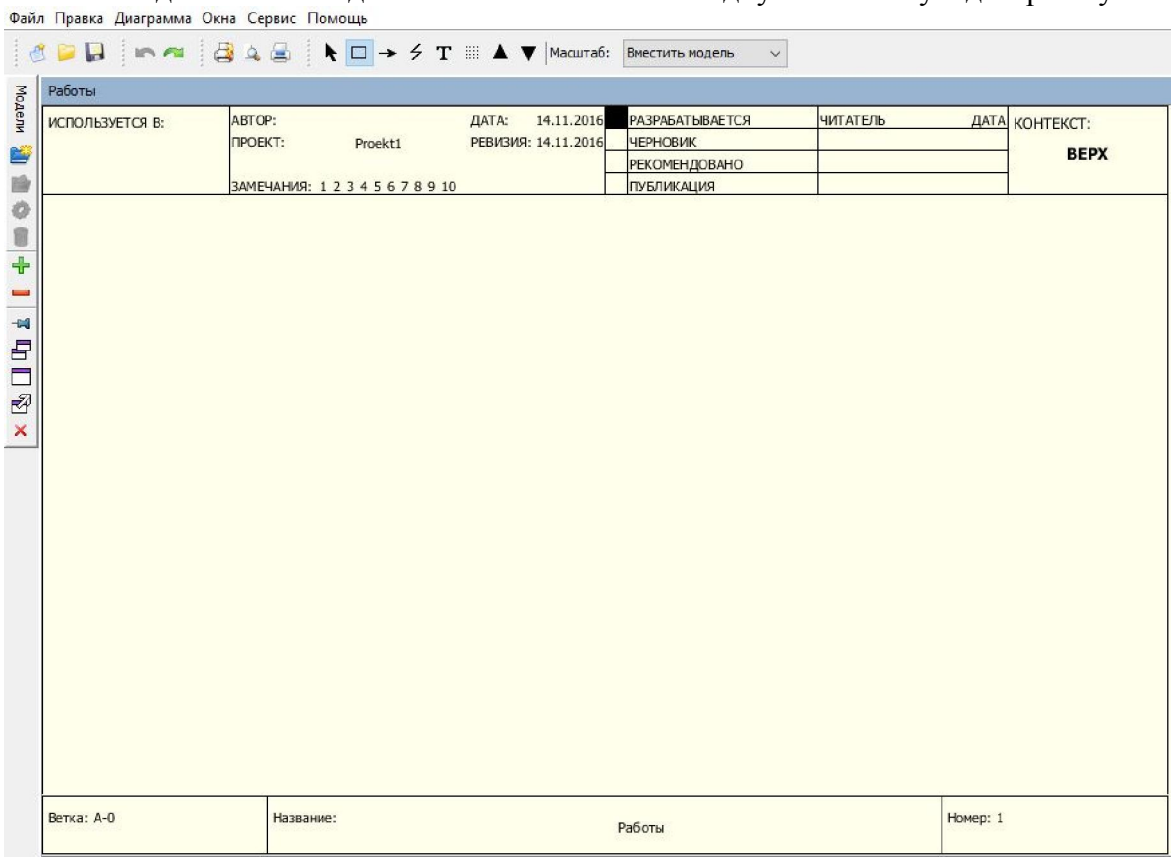


Рисунок 2.4 – Главное окно Ramus

Прежде, чем создавать остальные блоки, желательно сразу настроить среду проектирования, так как в противном случае в дальнейшем придется устанавливать настройки для каждого созданного блока в отдельности. Для этого в меню «Сервис»,

пункте «Свойства программы» на вкладке «Диаграмма» необходимо настроить все шрифты, которые будут действовать по умолчанию (рис. 1.5).

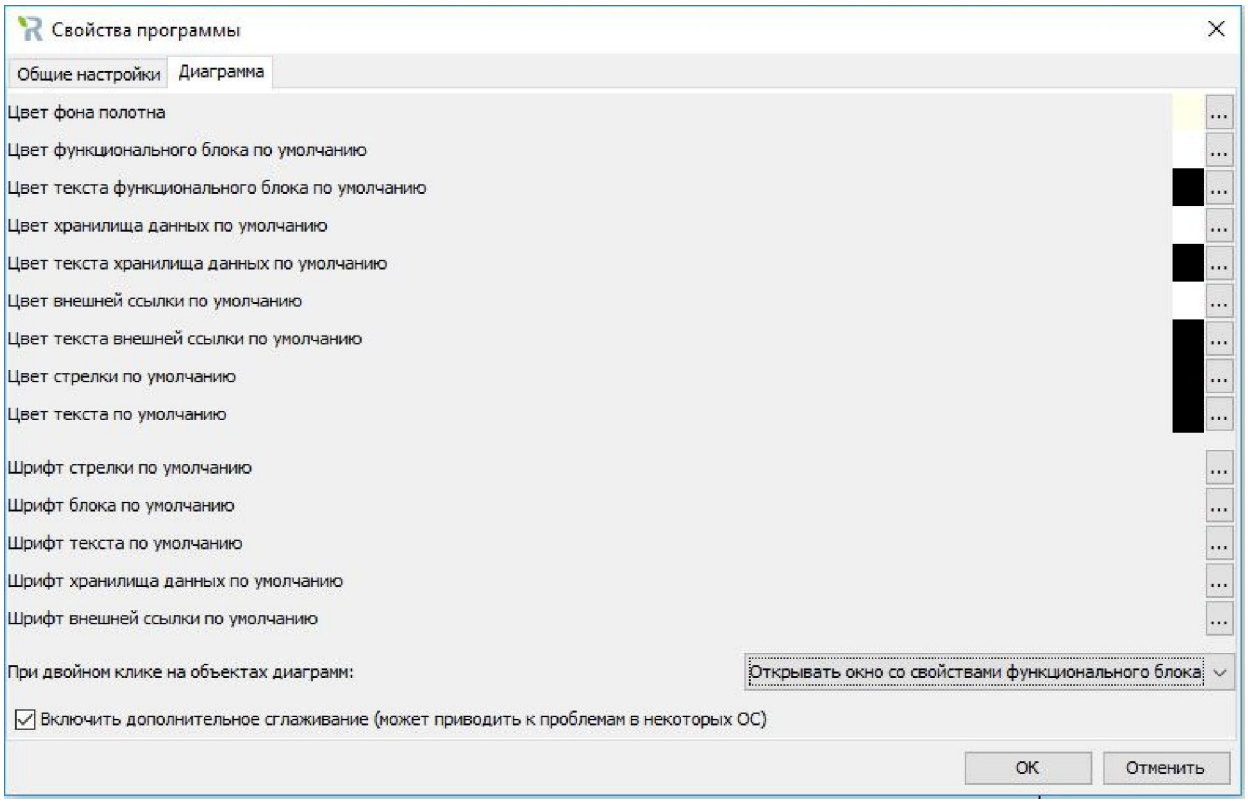


Рисунок 2.5 – Диалоговое окно «Свойства программы»

Для печати настройки устанавливаются в меню «Файл», пункте «Параметры страницы». Следует установить альбомное расположение листа, а также соответствующий формат. В настройках атрибутов проекта (Сервис→Свойства проекта→Автоматическое добавление атрибутов) устанавливаются соответствующие атрибуты.

Для добавления блока необходимо перейти в нужный режим путем вызова пункта «Режим добавления функциональных блоков» в меню «Диаграмма» или нажатием соответствующей кнопки на панели инструментов  . После выбора пункта или нажатия кнопки следует указать место, где будет расположен новый блок. Начальная нумерация для блока контекстной диаграммы – «A0».

Размеры новых блоков устанавливаются по умолчанию, но их можно изменить путем растягивания за его границы. Каждый блок в диаграмме должен быть подписан. Правильным является именование блоков с помощью глаголов неопределенной формы. До начала ввода подписи можно заранее установить нужные параметры шрифта в меню «Сервис», пункте «Свойства программы» на вкладке «Диаграмма», либо изменить параметры во вкладке «Шрифт» диалогового окна «Свойства функционального блока», которое вызывается двойным щелчком по самому блоку. Для того чтобы назвать блок, требуется выполнить одно из указанных действий:

- щелкнуть по функциональному блоку два раза и дать имя блоку во вкладке «Название»;
- щелкнуть правой кнопкой мыши по функциональному блоку и выбрать команду «Переименовать».

К функциональному блоку с четырех сторон должны быть присоединены различные объекты. Для ввода названий (ярлыков) объектов используется пункт «Режим добавления

текста» меню «Диаграмма», клавиша B клавиатуры или кнопка T на панели инструментов. После нажатия следует указать место вставки ярлыка и ввести нужный текст так же, как и для функционального блока. Отмена режима ввода надписей происходит при нажатии на клавишу Esc или на любую кнопку на панели инструментов. Редактировать текст можно двойным кликом по нему, в появившемся диалогом окне «Параметры текста» можно изменить его шрифт и цвет.

Стрелки на функциональной диаграмме предназначены для соединения ярлыков с функциональными блоками и блоков между собой. Чтобы появилась возможность нарисовать стрелку можно воспользоваться одним из предложенных вариантов:

- выбрать команду «Режим работы со стрелками» пункта меню «Диаграмма»;
- нажать на кнопку → на панели инструментов.

Далее нужно поместить курсор на границах модели до появления черной полосы (если стрелка идет извне в блок), либо у правой границы внутри функционального блока до появления черной стрелки (если стрелка идет из блока вовне). После чего протянуть курсор до появления черной стрелки внутри функционального блока, либо черной полосы у правой границы модели соответственно.

Стрелки на диаграмме могут сливаться в одну или наоборот разъединяться. Если несколько блоков должны быть соединены стрелками от одного источника, необходимо:

7 нарисовать стрелку к одному из блоков;
8 выделить ее в режиме работы со стрелками;
9 провести стрелку к нужному блоку.

Если несколько дуг от разных блоков должны соединяться в одну, необходимо:

10 нарисовать стрелку к одному из блоков;
11 выделить ее в режиме работы со стрелками;
12 провести следующую стрелку от блока к первой нарисованной стрелке (при наведении курсора на нее, она выделяется черным), две стрелки соединятся в одну.

Иногда на диаграмме требуется подписать стрелку, соединяющую функциональные блоки. Эта процедура выполняется в два этапа:

13 задать название стрелки, дважды щелкнув на нее и введя название в открывшемся диалоговом окне «Параметры стрелки» во вкладке «Поток»;
14 активизировать «Режим размещения тильд» в меню «Диаграмма», нажатием кнопки V на клавиатуре или на панели инструментов, нажать на название стрелки и подвести курсор мыши к стрелке, которую требуется подписать. При этом стрелка выделится черным, после чего нажать на нее. По окончании процедуры от подписи к стрелке будет проведена волнистая линия, символизирующая их взаимную связь.

Для создания следующего функционального уровня модели необходимо пометить щелчком мыши функциональный блок и выполнить команду «Перейти к дочерним диаграммам» пункта меню «Диаграмма», либо выбрать кнопку на панели инструментов ▼. Появится диалоговое окно, в котором следует выбрать шаблон уровня и количество функциональных блоков (рис. 1.6).

На данной странице модели уже находятся ярлыки, созданные на предыдущем уровне. Для перехода на родительскую или дочернюю диаграммы используются пункты «Перейти к родительской диаграмме» и «Перейти к дочерним диаграммам» меню «Диаграмма» или кнопки ▲ и ▼ на панели инструментов.

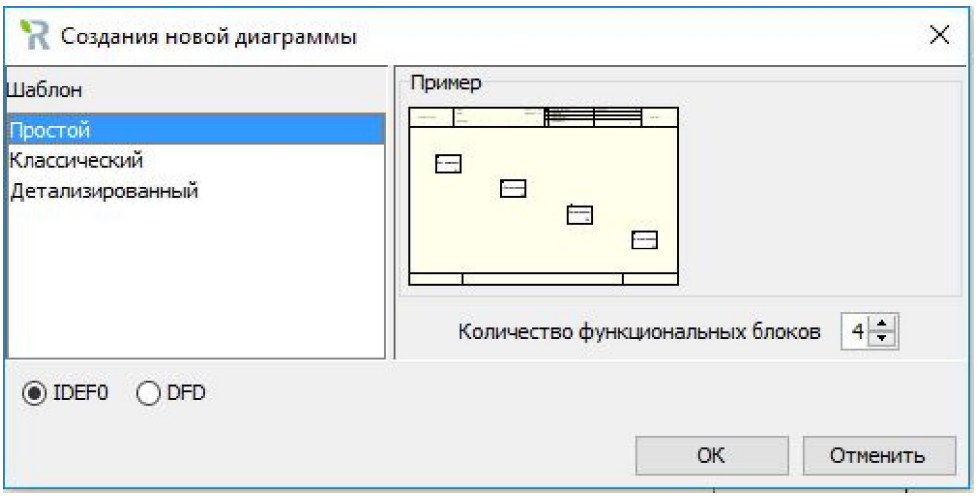


Рисунок 2.6 – Диалоговое окно «Создания новой диаграммы»

В программе предусмотрена возможность упорядоченного хранения информации о таких объектах, как персонал, документы, функции и т.д. в виде системы классификаторов. Классификация объектов значительно упрощает поиск и обработку информации об объектах модели. Каждый элемент системы классификации, кроме названия, может иметь дополнительные атрибуты, в которых можно упорядочено хранить разнообразнейшую информацию об объекте.

Заполнение классификатора производится во вкладке «Классификаторы» в правом верхнем углу модели либо нажатием «Окна» – «Рабочее пространство» – «Классификаторы». Добавление нового классификатора происходит путем нажатия кнопки  в окне «Классификаторы» в верхнем левом углу (рис. 2.7). Каждый классификатор должен содержать элементы, которые добавляются в центральном окне рабочего пространства.

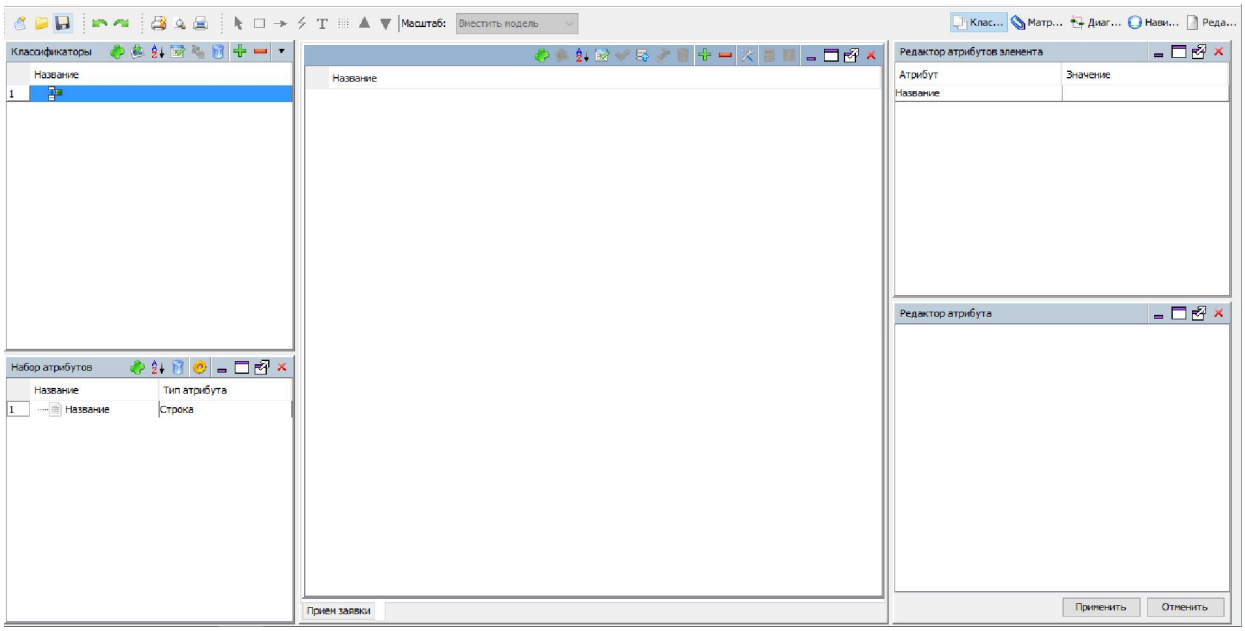


Рисунок 2.7 – Рабочее пространство «Классификаторы»

Элементы классификатора можно сортировать по названию, объединять и перемещать в другой классификатор с помощью соответствующий кнопок на панели инструментов. Также можно группировать элементы внутри классификатора, для этого

необходимо нажать на номер элемента и, не отпуская курсор, перенести его на другой элемент. Появится папка, содержащая перенесенный элемент классификатора.

Классификаторы могут иметь разный набор атрибутов. Новый атрибут добавляется нажатием кнопки  на панели инструментов окна «Набор атрибутов» в левом нижнем углу рабочего пространства. Появится окно «Добавление атрибута» (рис. 2.8), в нем необходимо выбрать тип атрибута и классификаторы, к которым он будет применяться.

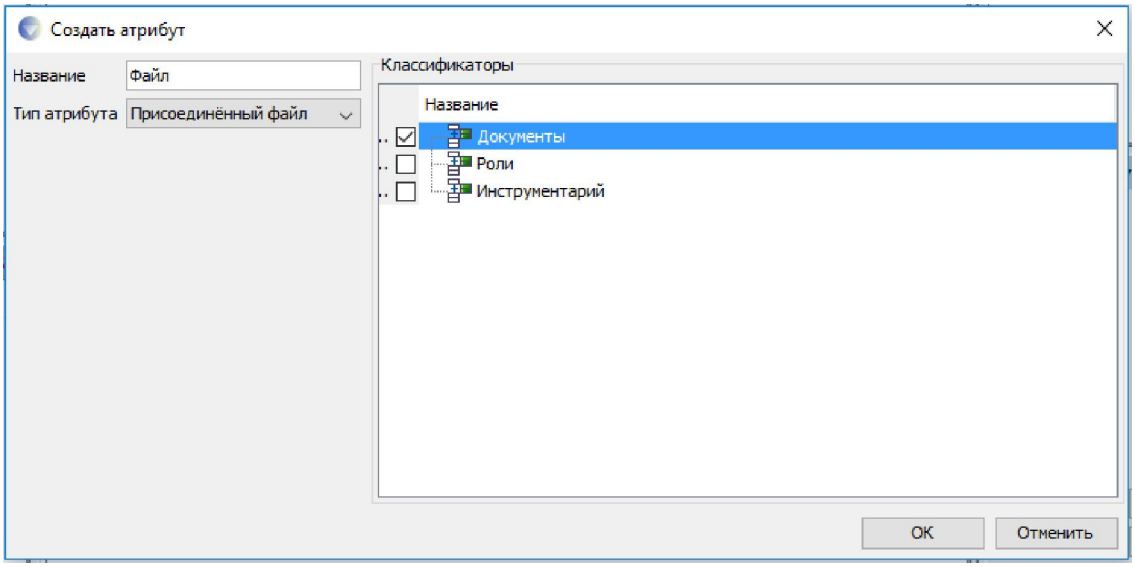


Рисунок 2.8 – Диалоговое окно «Создать атрибут»

Таким образом, появляется возможность дополнять элемент различной информацией (рис. 2.9). Рассмотрим, какие еще типы атрибутов можно использовать. Для этого надо отредактировать атрибут с помощью соответствующей кнопки на панели инструментов. Для удаления атрибута он не должен использоваться в классификаторах, для этого нужно убрать флажок классификатора в «Свойствах атрибута», после чего появится возможность его удалить.

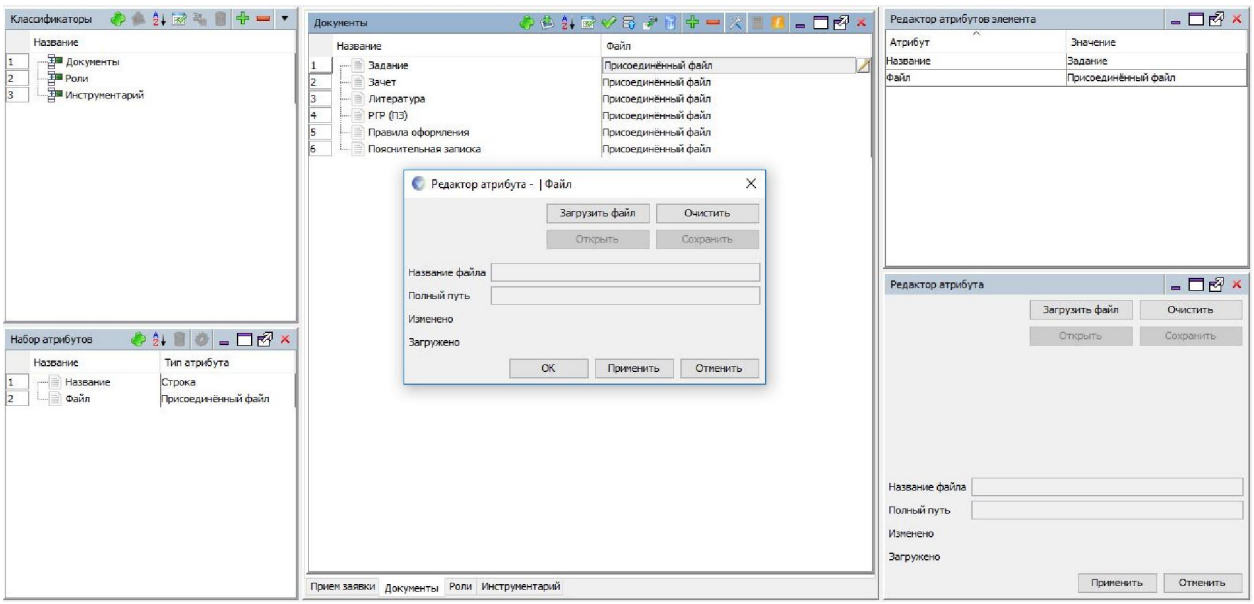


Рисунок 2.9 – Диалоговое окно «Редактор атрибута»

Также в программе имеется возможность присоединять элементы классификатора к названиям стрелок. Для этого необходимо открыть диалоговое окно «Параметры стрелки» в рабочем пространстве «Диаграммы», дважды кликнув по стрелке. Для присоединения

элементов классификатора нужно нажать кнопку «Добавить» и выбрать элемент классификатора или весь классификатор, если это необходимо (рис. 2.10). Можно присоединять несколько элементов классификатора, тогда они будут указаны на диаграмме через точку с запятой. Удалить их можно с помощью кнопки «Удалить».

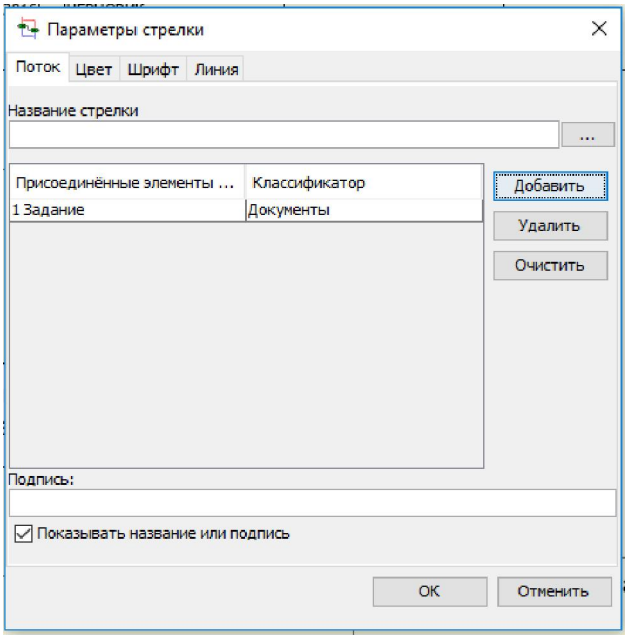


Рисунок 2.10 – Диалоговое окно «Параметры стрелки»

IDEF0-модель сохраняется на диске с помощью команд «Файл»→«Сохранить» или «Файл»→«Сохранить как…».

Далее приведен пример IDEF0 модели «Выполнить расчетно-графическую работу (РГР)» (рис. 2.11, 2.12).

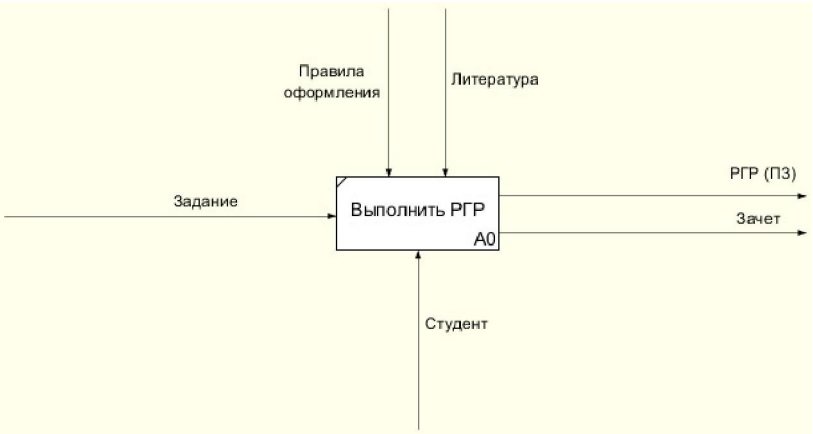


Рисунок 2.11 – Первый уровень функциональной модели «Выполнить РГР»

Элементы классификатора функциональной модели «Выполнить РГР»:

- Задание – постановка задачи на выполнение РГР.
- Зачет – оценка в баллах, выставляемая студенту за РГР.
- Литература – список литературы для самостоятельной проработки в рамках РГР.
- Метод решения – метод решения поставленной задачи.
- План, график работы – план выполнения РГР.
- Пояснительная записка – оформленная в соответствии с требованиями пояснительная записка к РГР.
- Правила оформления – методические указания по выполнению и оформлению РГР.
- РГР (ПЗ) – расчетно-графическая работа (пояснительная записка).
- Расчеты – расчеты, необходимые для решения поставленной задачи.

- Студент – учащийся ВУЗа.

Иерархическое представление функциональной модели «Выполнить РГР»:

[A0] Выполнить РГР

[A1] Составить план работы

[A2] Изучить литературу и выбрать метод (методику, приемы решения)

[A3] Выполнить необходимые расчеты на ЭВМ

[A4] Подготовить пояснительную записку

[A5] Защитить РГР

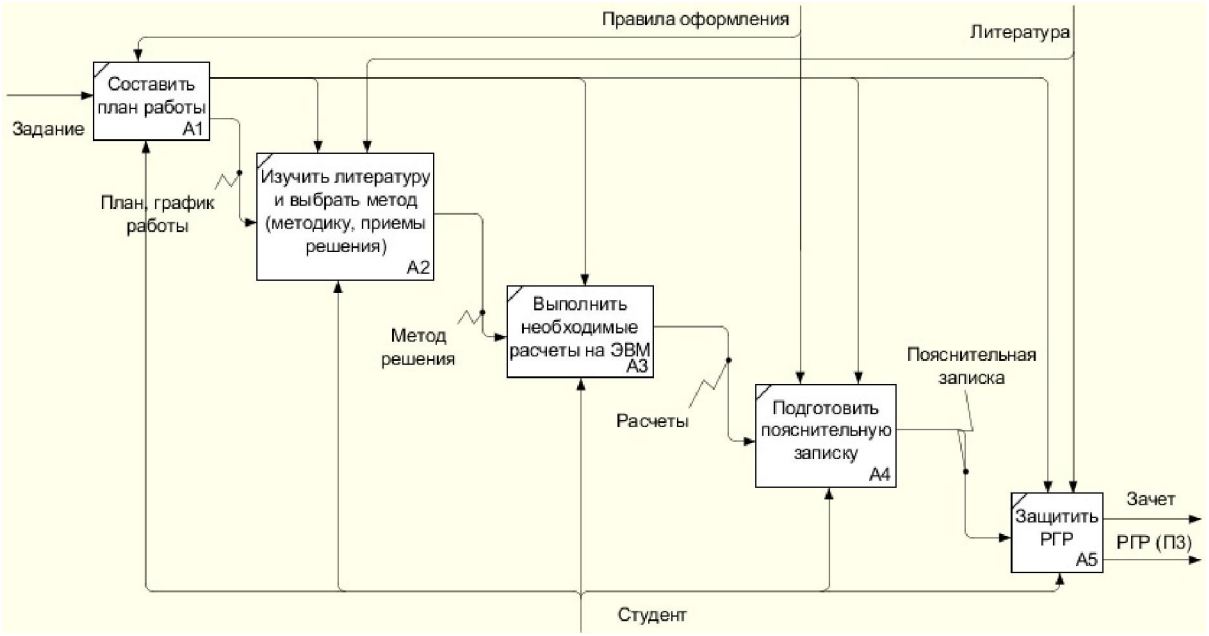


Рисунок 2.12 – Декомпозиция блока A0 функциональной модели «Выполнить РГР»

2.2 Создание диаграммы корневого и первого уровней: выполнение работы на примере «Создание продукта»

Цель работы – создать диаграммы корневого и первого уровня на примере модели «Создание продукта», используя специальные средства и команды Ramus (создание IDEF-блоков и дуг, создание текста и т.д.).

После запуска программы в открывшемся диалоговом окне нужно выбрать «Создать новый файл», поставьте галочку IDEF0, щелкнуть OK. В открывшемся окне указать название проекта и нажать «Окончить». Переключать окна можно через меню Window. Для сохранения модели нужно выбрать меню «Файл» – «Сохранить как…» и в открывшемся диалоговом окне ввести имя файла, рекомендуется использовать свою фамилию (расширение rsf автоматически добавится к имени файла).

В Ramus термин страница – синоним термина диаграмма. При создании новой модели автоматически создаётся диаграмма верхнего уровня (контекстная диаграмма) A-0. Для создания блока A0 необходимо перейти в нужный режим путем вызова пункта «Режим добавления функциональных блоков» в меню «Диаграмма» или нажатием соответствующей кнопки на панели инструментов  и щелкнуть там, где будет располагаться блок.

Для ввода названия блока A0 необходимо дважды кликнуть по нему и в появившемся диалоговом окне во вкладке «Название» ввести «Создание продукта».

Изменить шрифт, размер, начертание, цвет текста можно в соответствующих вкладках диалогового окна «Свойства функционального блока». Блок A0 можно перемещать с помощью мыши, изменять размеры (потянув за границы блока), удалить (с помощью клавиши Delete).

Стандартная контекстная IDEF0-диаграмма включает формулировки цели и точки зрения модели. В Ramus это делается в виде метки. Для этого нужно выбрать в меню «Диаграмма» пункт «Режим добавления текста» или щелкнуть по кнопке T на специальной панели инструментов. После этого, щелкнув левой кнопкой мыши ниже блока A0, и в появившемся тексте «(Без названия)», дважды щелкнуть по нему и во вкладке «Текст» открывшегося окна «Параметры текста» ввести: «Цель: Организовать процесс создания продукта на заказ». Далее нажав Enter ввести: «Точка зрения: Группа разработки». После этого нажать ОК.

После этого метку с помощью мыши можно передвинуть в нижнюю часть страницы. Если требуется изменить текст метки, то для этого нужно дважды щелкнуть по ней мышью и отредактировать текст в открывшемся окне.

Чтобы нарисовать стрелку нужно выбрать команды меню «Диаграмма» – «Режим работы со стрелками» или щелкнуть по кнопке → на панели инструментов. Поместить курсор на границах модели до появления черной полосы (если стрелка идет извне в блок), либо у правой границы внутри функционального блока до появления черной стрелки (если стрелка идет из блока вовне) и протянуть курсор до появления черной стрелки внутри функционального блока, либо черной полосы у правой границы модели соответственно. Требуется создать и разметить стрелки, как показано на рис. 2.13. Чтобы, центрировать стрелки, нужно нажать на блок правой кнопкой мыши и выбрать пункт «Центрировать присоединенные стрелки».

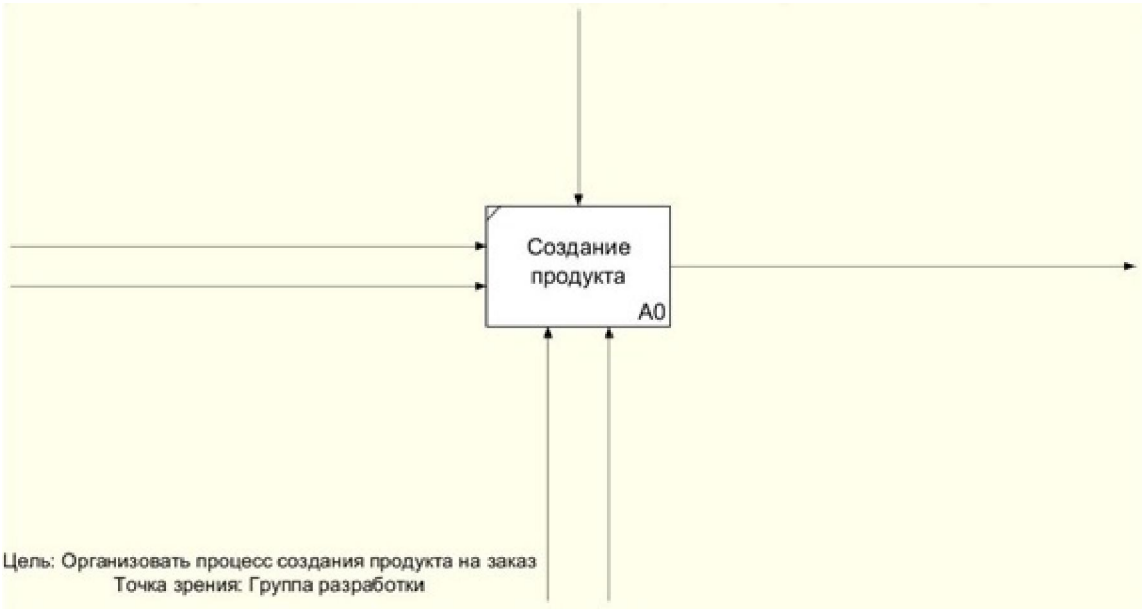


Рисунок 2.13 – Размещение стрелок

Далее требуется именовать стрелки. Рассмотрим процесс именования стрелки управления. Для этого нужно дважды нажать на верхнюю стрелку и в открывшемся окне «Параметры стрелки» ввести её название: «Спецификации». Также если необходимо можно изменить шрифт и цвет и затем нажать ОК. После этого текст можно передвинуть на требуемое место. Отмена режим работы со стрелками осуществляется нажатием Esc.

Именование остальных стрелок осуществляется аналогично, рис. 2.14. Выбрав «Файл» – «Сохранить», сохранить изменения.

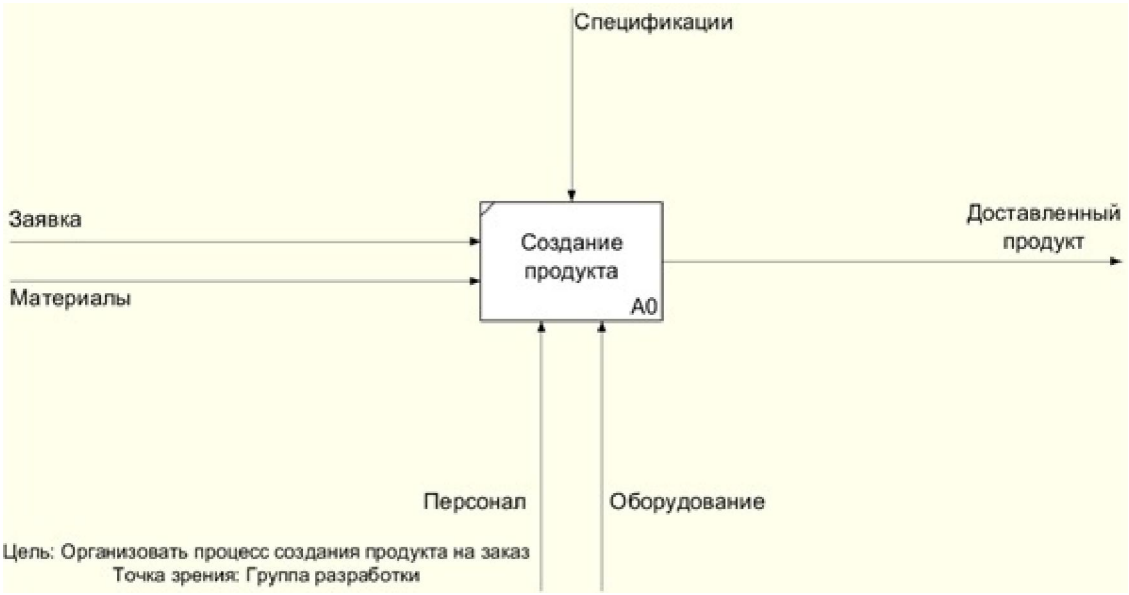


Рисунок 2.14 – Создание меток

Чтобы создать новую дочернюю диаграмму, содержащую функциональные блоки, детализирующие содержание родительского блока A0, необходимо этот блок декомпозировать. Для этого выбрать блок A0, щелкнув по нему мышью. Далее выбрать команду меню «Диаграмма» – «Перейти к дочерним диаграммам» или щелкнуть по кнопке ▼ на специальной панели инструментов. В открывшемся окне «Создание новой диаграммы» выбрать шаблон «Простой», количество блоков: 3 и нажать ОК. Будет создана новая страница (номер 2) для представления диаграммы следующего уровня.

Вернуться на родительскую диаграмму (страница 1) можно, выбрав команду меню «Диаграмма» – «Перейти к родительской диаграмме» или щелкнув по кнопке ▲. Атрибуты страницы 2 будут автоматически заполнены. Текст меток («портовых узлов») блока A0 переносится на созданную страницу по ее краям соответственно расположению на родительской диаграмме. Изменять масштаб можно с помощью выпадающего списка «Масштаб» на панели инструментов. Для удобства рекомендуется установить масштаб «Вместить модель».

Диаграмма первого уровня будет содержать три функциональных блока, детализирующих блок A0 «Создание продукта»:

- A1 «Прием заявки».
- A2 «Изготовление продукта».
- A3 «Доставка продукта».

Соответственно сначала требуется именовать существующие блоки. Для этого дважды щелкнув по блоку A1 во вкладке «Название» ввести «Прием заявки». Аналогичным образом именуются блок A2 («Изготовление продукта») и A3 («Доставка продукта»). Если текст не помещается блок, можно увеличить ширину блока. Если на диаграмму требуется добавить еще один блок, то нужно выбрать команду меню «Диаграмма» – «Режим добавления функциональных блоков» или щелкнув по кнопке □ панели инструментов. Удалить блок можно, выделив его и нажав клавишу Delete.

Далее необходимо соединить портовые узлы, которые были перенесены с родительской диаграммы, с блоками. Для этого нужно выделить мышью входной

портовый узел «Заявка» и передвинуть его, чтобы он был расположен слева от блока А1. При этом метка «Заявка», связанная с этим узлом, тоже переместится. В режиме курсора (кнопка  на панели инструментов) щелкнуть на стрелку, она должна выделится черным, и провести курсор (за ним должна тянуться пунктирная линия) до блока А1, пока у правой границы блока не появится черная стрелка, по которой нужно щелкнуть. Стрелка портового узла «Заявка» соединится с функциональным блоком А1.

Аналогичным образом нужно создать оставшиеся дуги:

- от узла «Материалы» к левой стороне блока А2;
- от узла управления «Спецификации» – к верхней стороне блока А2;
- от узла механизма «Персонал» – к нижней стороне блока А1;
- от узла механизма «Оборудование» – к нижней стороне блока А2;
- от правой стороны блока А3 – к выходному узлу «Доставленный продукт».

Затем нужно добавить новый портовый узел, которого не было на родительской диаграмме: «Инструкции». Для этого нужно создать стрелку, идущею от верхней границы модели к верхней границы блока А1. Верхняя часть стрелки будет помещена в туннель в виде квадратных скобок: [ ]. Необходимо нажать на туннель правой кнопкой мыши и выбрать команду «Туннель», в открывшемся диалоговом окне включить параметр «Обозначить туннель круглыми скобками». Стрелка будет помещена в туннель в виде круглых скобок: ( ). Это означает, что стрелка идет от портового узла, который не был перенесен с родительской диаграммы. Далее нужно именовать стрелку, дважды щелкнув мышью на нее и ввести «Инструкции». Затем нажать «Диаграмма» – «Центрировать все стрелки». Результат проделанной работы приведен на рис. 2.15.

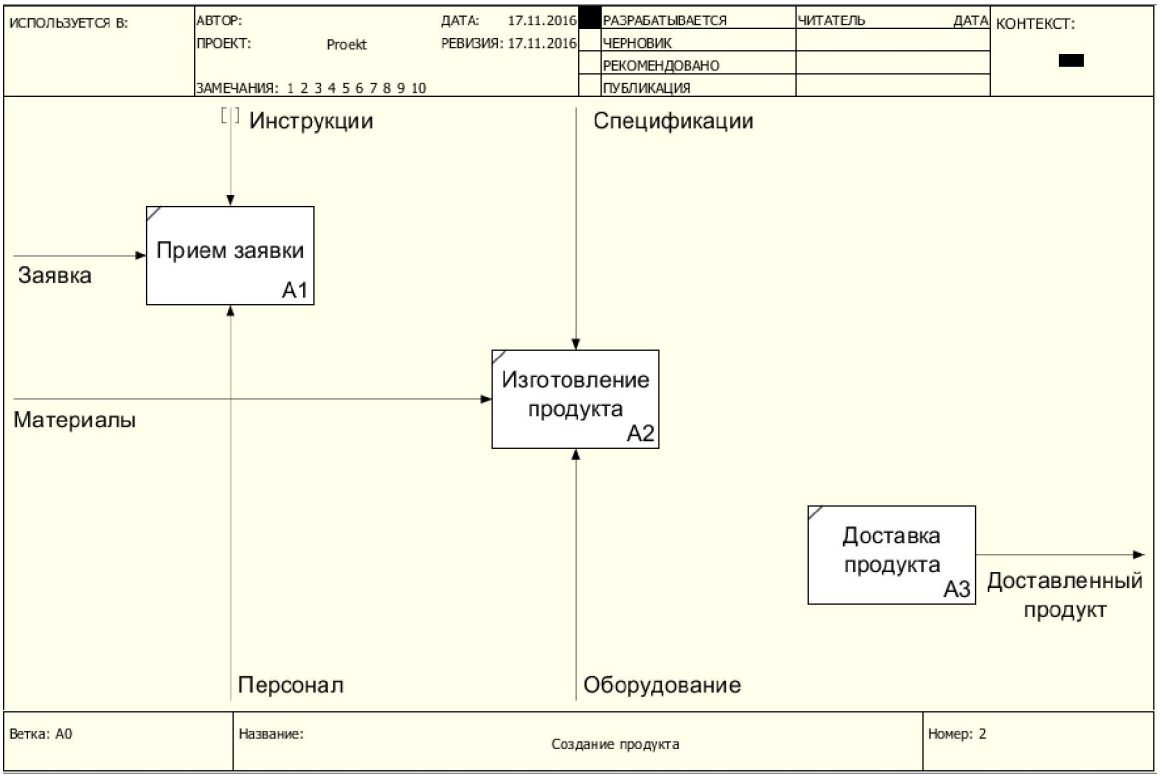


Рисунок 2.15 – Создание внешних стрелок на диаграмме первого уровня

После этого нужно соединить блоки друг с другом. Так, выход блока А1 «Заказ» должен являться для блока А2 управлением, т.к. описание продукта, содержащееся в заказе, показывает, как должно осуществляться изготовление продукта. Чтобы создать стрелку, являющуюся выходом для блока А1 и управлением для блока А2, нужно выбрать

команду меню «Диаграмма» – «Режим работы со стрелками» или соответствующую кнопку на специальной панели инструментов и соединить правую сторону блока А1 с верхней стороной блока А2. Если стрелка получилась неправильно (например, вошла не с той стороны блока, с которой нужно), её нужно выделить и удалить, после чего повторить процесс.

Выход блока А2 «Продукт» должен являться входом для блока А3. Чтобы пометить стрелки, соединяющие блоки А1, А2 и А3 друг с другом, нужно сначала создать для каждой стрелки присоединенную метку. Для этого нужно дважды щелкнуть по стрелке, идущей от блока А1 к блоку А2, и назначить ей название «Заказ». Нажать Esc. Далее перейти в меню «Диаграмма» – «Режим размещения тильд» или нажать кнопку  на панели инструментов. Затем поочередно нажать на метку «Заказ» и саму стрелку. Метка будет соединена со стрелкой линией. Если перемещать метку, она все равно будет соединена со стрелкой.

Аналогично описанному выше нужно создать метку «Продукт» правее вертикального сегмента стрелки, соединяющей блоки А2 и А3. Присоединить созданную метку к этому сегменту стрелки. Если метка присоединилась не в том месте стрелки, можно нажать на стрелку, к которой присоединена метка, в новом месте при включенном режиме размещения тильд. Требуется попробовать изменить место соединения метки и стрелки. Выбрав «Файл» – «Сохранить», сохранить изменения.

Самостоятельная работа

Выбрать бизнес-процесс, для которого будет разрабатываться IDEF0-модель. Список рекомендуемых процессов приведен в приложении. По согласованию с преподавателем можно выбрать процесс не из списка.

Далее требуется собрать информацию о выбранном бизнес-процессе, в том числе:

- об основных этапах выполнения процесса;
- о результатах выполнения каждого этапа и о том, как используются эти результаты;
- о материалах, необходимых для выполнения каждого этапа, и источниках их получения;
- об исполнителях каждого этапа и об оборудовании, используемом при его выполнении.

2.3 Создание диаграмм декомпозиции: выполнение работы на примере «Создание продукта»

Цель работы – закончить создание диаграммы первого уровня модели «Создание продукта» и создать диаграмму декомпозиции на базе диаграммы первого уровня.

Сначала требуется открыть модель «Создание продукта», созданную на предыдущем занятии, выбрав команду при запуске программы «Открыть файл», либо после запуска программы «Файл» – «Открыть» и выбрав имя файла.

Выход блока А1 «Заказ» должен являться управлением не только для блока А2, но и для блока А3, т.к. адрес клиента, содержащийся в заказе, в некотором роде управляет выполнением блока А3 (показывает, куда осуществляется доставка продукта). Для того чтобы создать разветвление:

- выбрать режим работы со стрелками;
- выделить стрелку, соединяющую блоки А1 и А2;

- протянуть указатель мыши на верхнюю сторону блока А3 и, когда у верхней границы блока появится черная стрелка, щелкнуть левой кнопкой мыши по ней.

Ветвь стрелки будет проведена.

Теперь необходимо разветвить стрелку от портового узла «Персонал», т.к. он является механизмом не только для блока А1, но и для блоков А2 и А3. Для этого нужно аналогично создать ветви стрелки, соединяющей «Персонал» с блоками А2 и А3. При этом стоит помнить, что местом присоединения ветвей должны быть нижние стороны блоков (указатель мыши нужно помещать на нижние стороны блоков). Требуется присоединить метки: к ветви, являющейся механизмом блока А1 – метку «Отдел приема заявок»; к ветви, являющейся механизмом блока А2 – метку «Цех»; к ветви, являющейся механизмом блока А3 – метку «Отдел доставки». Результат проделанной работы приведен на рис. 2.16.

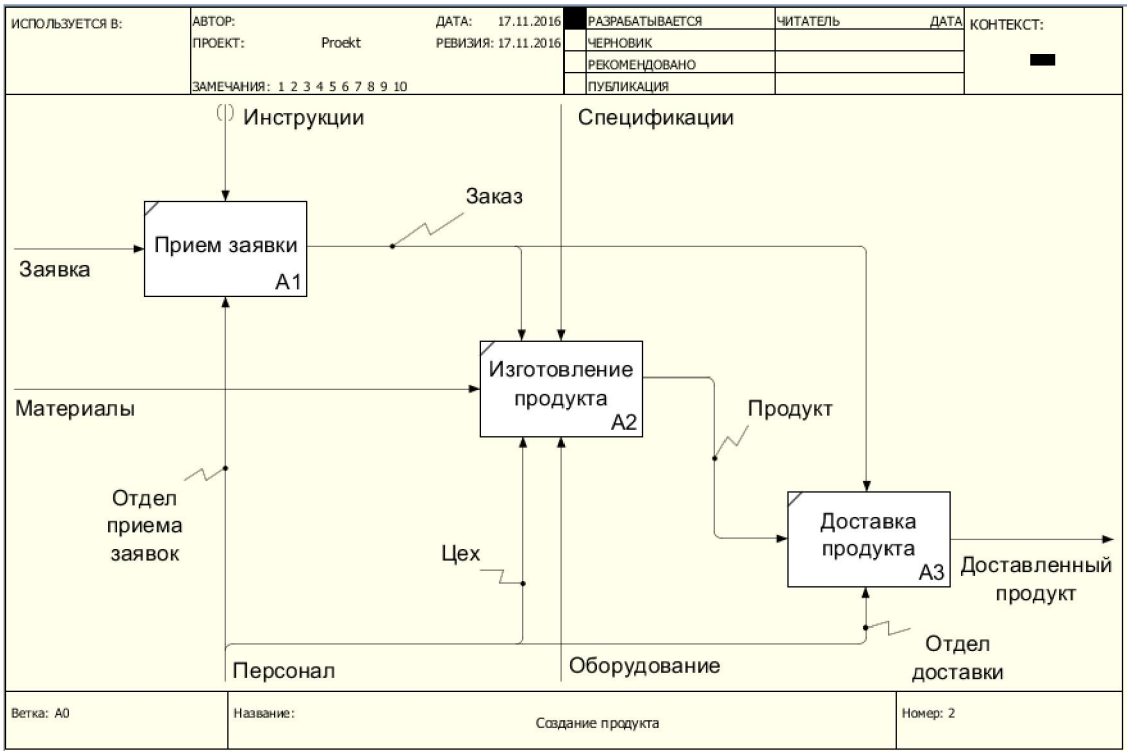


Рисунок 2.16 – Ветвление дуг

Для дальнейшей детализации блока А1 «Прием заявки» его нужно декомпозировать. Для этого нужно создать дочернюю диаграмму блока А1 аналогично тому, как была создана диаграмма первого уровня (см. предыдущею работу). На дочерней диаграмме блока А1 нужно разместить три блока: А11 «Выбор продукта», А12 «Оформление заказа» и А13 «Оплата». Размещение блоков было описано в предыдущей работе.

На дочерней диаграмме не должно быть метки «Инструкции», поэтому её нужно выделить и удалите. Затем нужно вернуться на страницу 2, на диаграмме появится еще один туннель у стрелки «Инструкции». Нужно задать ему круглые скобки. Этот туннель означает, что управление «Инструкции» не будет переноситься на дочерние диаграммы.

Если блоки менять местами, то они будут перенумеровываться автоматически.

Далее разместить на диаграмме метки и нарисовать стрелки так, чтобы получилось что-то похожее на рис. 2.17.

Для перемещения стрелки требуется:

- выделить любую стрелку, щелкнув по ней два раза;
- не отпуская указатель, переместить стрелку в нужное место.

Стрелка выхода «Заказ» должна являться не только выходом блока A13, но и выходом блока A12, т.е. должна сливаться из выходов двух блоков. Для соединения (слияния) стрелок требуется:

- перейти в режим работы со стрелками;
- нажать на правую границу блока A12 и тянуть указатель до середины стрелки, выходящей из блока A13;
- должна получиться стрелка выхода из A12, сливающаяся с выходом из блока A13.

Выбрать «Файл» – «Сохранить», чтобы сохранить изменения.

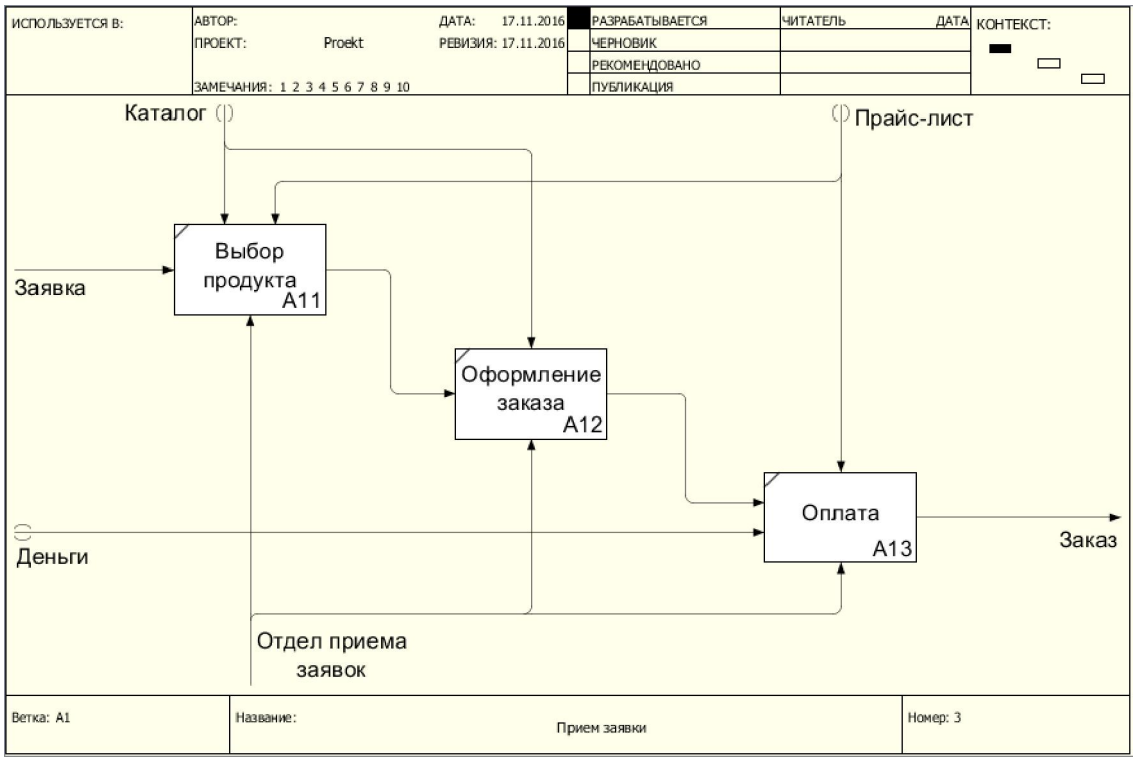


Рисунок 2.17 – Диаграмма декомпозиции блока A1

Самостоятельная работа

Требуется собрать информацию о каждом из основных этапов выбранного бизнес-процесса, в том числе:

- о последовательности работ в ходе выполнения этапа;
- о результатах выполнения каждой работы и о том, как используются эти результаты;
- о материалах, необходимых для выполнения каждой работы, и источниках их получения;
- об исполнителях каждой работы и об оборудовании, используемом при ее выполнении;
- об управляющей информации, необходимой для выполнения каждой работы.

После этого нужно представить собранную информацию в виде текстового описания и составить рукописные IDEF0-диаграммы.

Далее нужно открыть файл, в котором хранится модель, созданная на предыдущей занятии по индивидуальному заданию и в ней создать диаграммы декомпозиции для каждого из основных блоков диаграммы первого уровня.

2.4 Создание словаря данных: выполнение работы на примере «Создание продукта»

Цель работы – создать словарь данных для IDEF0-модели «Создание продукта», в котором будет храниться информация о графических объектах модели.

Открыть модель «Создание продукта», созданную на предыдущем занятии, выбрав команду меню «Файл» – «Открыть» и выбрав имя файла. Перейти в рабочее пространство «Классификаторы». В работе будет создано два типа классификатора: Функция и Кадры. В классификаторе «Кадры» будут содержаться элементы для описания подразделений персонала, таких как «Отдел приема заявок», «Цех» и «Отдел доставки».

После их создания, для классификатора «Кадры» нужно создать атрибуты, в которых будет храниться информация о подразделении (руководитель, количество сотрудников, списочный состав, место расположения, телефон). Для этого в окне «Набор атрибутов» нажать кнопку  и в появившемся диалоговом окне «Создать атрибут» дать присвоить название атрибуту и соответствующий тип и потом поставьте флажок у классификатора «Кадры». Аналогично требуется создать следующие атрибуты для данного классификатора:

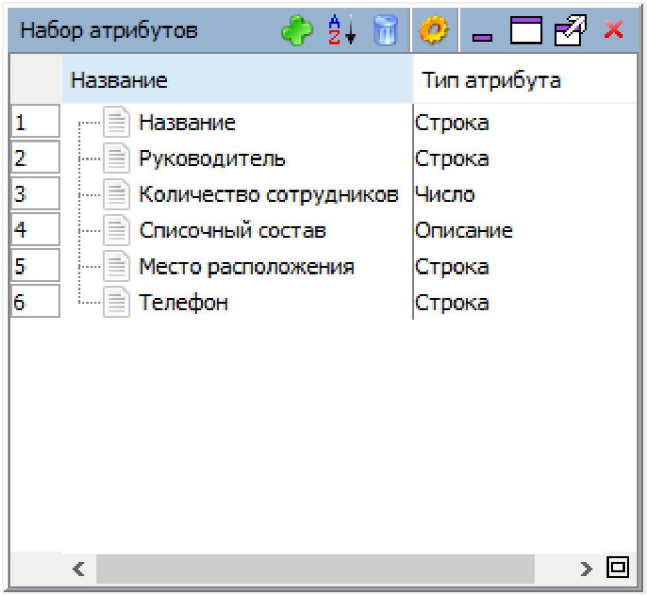


Рисунок 2.18 – Окно «Набор атрибутов»

После этого в классификаторе «Кадры» появятся дополнительные поля: Руководитель, Количество сотрудников, Списочный состав, Место расположения, Телефон (рис. 2.19). Их нужно заполнить для каждого отдела. Нужно обратить внимание на то, чтобы заполнить «Списочный состав» с типом атрибута: описание, необходимо нажать на кнопку справа от описания 

или ввести текст в окне «Редактор атрибута».

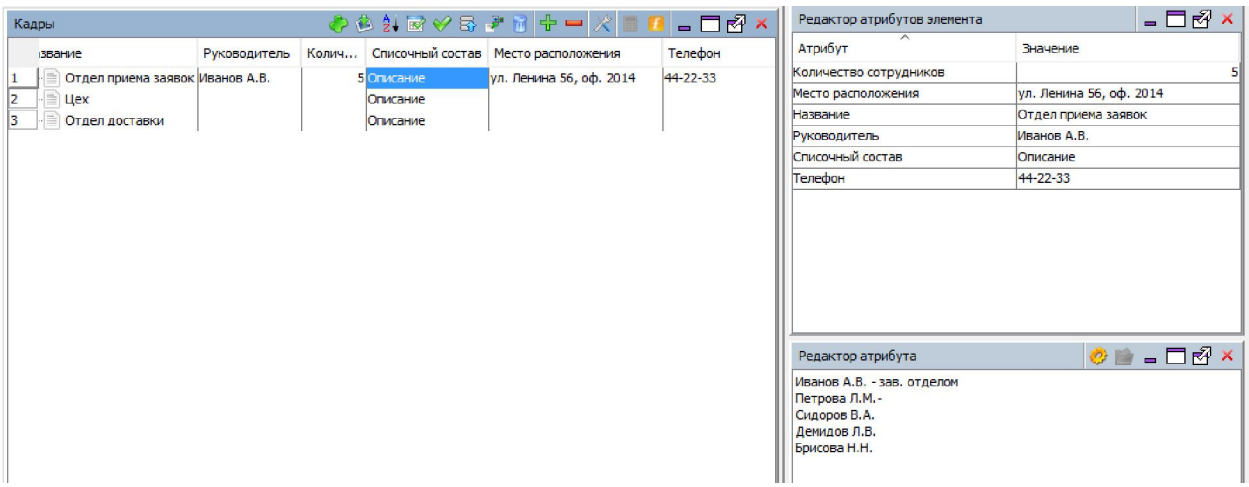


Рисунок 2.19 – Рабочее пространство «Классификаторы»

После этого нужно перейти к созданию классификатора «Функции». Для этого в центральном окне рабочего пространства нужно создать элементы классификатора в соответствии с названиями функций на диаграмме: «Прием заявки», «Изготовление продукта» и «Доставка продукта». Этот классификатор будет использоваться для описания функциональных блоков модели, и будет содержать информацию о соответствующей функции (описание функции, средняя продолжительность ее выполнения, сотрудники, выполняющие функцию).

Аналогично созданию атрибутов для предыдущего классификатора, требуется создать три атрибута, не забывая при создании ставить флажок для классификатора «Функции»:

- описание – тип: описание;
- продолжительность (час) – тип: число;
- сотрудники – тип: элемент классификатора (установите классификатор «Кадры», созданный Вами раннее и его атрибут «Название»).

После этого нужно заполните появившиеся поля. Для «Приема заявок» в поле «Описание» ввести, например, такой текст: «Консультирование клиента. Оформление заказа. Прием оплаты.», ввести значение для поля «Продолжительность (час)» - 0.5. Следует обратить внимание на то, что в поле «Сотрудники» появляется выпадающий список, в котором необходимо выбрать соответственно «Отдел приема заявок» (рис. 2.20). Аналогично заполняются оставшиеся элементы классификатора.

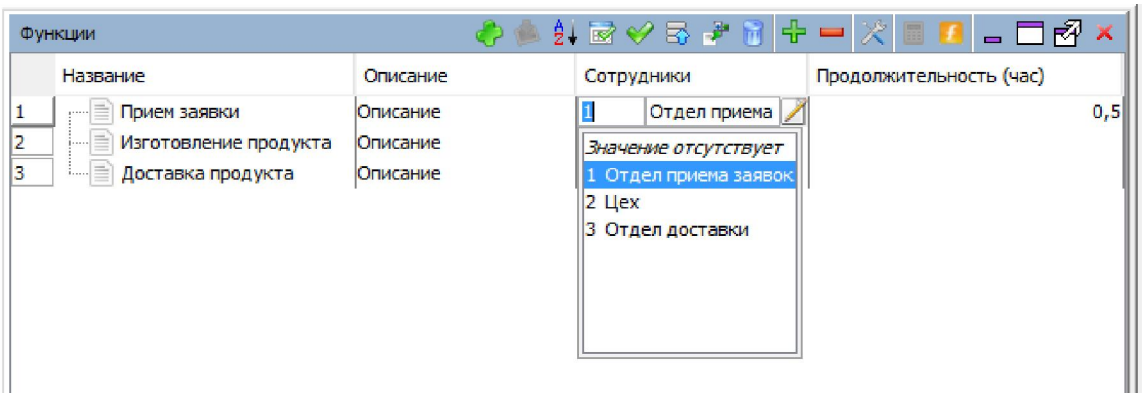


Рисунок 2.20 – Окно задания значений полей записи

Для обновления номера версии модели нужно выбрать команду меню «Файл» – «Сохранить».

Созданные классификаторы можно экспортировать и импортировать. Для экспорта выбрать «Файл» – «Экспортировать классификаторы». Выбрав путь сохранения, в появившемся диалоговом окне «Классификаторы», нужно указать те, которые необходимо экспортировать, после чего нажать ОК. Классификаторы сохраняются в том же формате .rsf, что и проект.

Самостоятельная работа

Для выбранного в качестве индивидуального задания бизнес-процесса нужно выделить основные типы объектов, информация о которых будет содержаться в словаре. Это могут быть:

- подразделения организации, участвующие в выполнении бизнес-процесса;
- документы (планы, заказы, проекты), создаваемые в ходе выполнения процесса;
- продукция (услуги), создаваемые в ходе выполнения процесса;
- этапы (подэтапы, работы) процесса.

Для каждого выделенного типа объектов требуется составить список атрибутов (полей данных), достаточных для описания данного типа. Также нужно определить какого типа должны быть атрибуты.

Далее для каждого типа объектов нужно составить перечень конкретных объектов данного типа (содержащихся в модели в качестве меток или блоков). Затем требуется определить какими должны быть конкретные значения атрибутов для каждого из выделенных объектов.

После этого нужно открыть файл, в котором хранится модель, созданная на предыдущем занятии по индивидуальному заданию. Требуется определить типы записей. Каждому типу объектов должен соответствовать свой тип записей. При определении типа записи каждому атрибуту сопоставляется поле данных. С учетом этого нужно создать записи для конкретных объектов, содержащихся в модели в качестве меток или блоков. При создании записи ввести значения атрибутов в соответствующие поля данных. После этого требуется сохранить словарь.

2.5 Дополнение созданной модели процессов DFD-диаграммами: выполнение работы на примере «Создание продукта»

Цель работы: создать DFD-диаграмму на примере модели «Создание продукта».

Диаграммы потоков данных (DFD) – представляют собой сеть связанных между собой работ. Их удобно использовать для описания документооборота и обработки информации. DFD описывает:

- функции обработки информации – работы;
- документы, объекты, сотрудников или отделы, участвующие в процессе обработки информации;
- внешние ссылки, которые обеспечивают интерфейс с внешними объектами, находящимися за границами моделируемой системы;
- таблицы для хранения документов (хранилища данных).

Для построения DFD-диаграммы используются поток данных, процесс, хранилище и внешняя сущность.

Потоки данных являются механизмами, использующимися для моделирования передачи информации (или физических компонентов) из одной части системы в другую. Потоки изображаются на схеме именованными стрелками, ориентация которых указывает

направление движения информации. Стрелки могут подходить к любой грани работы и могут быть двунаправленными для описания взаимодействия типа команда-ответ.

Назначение процесса состоит в продуцировании выходных потоков из входных, в соответствии с действием, задаваемым именем процесса. Каждый процесс должен иметь уникальный номер для ссылок на него внутри диаграммы.

Хранилище данных позволяет на определенных участках определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилища – это «срезы» потоков данных во времени. Информация, содержащаяся в хранилище, может использоваться в любое время после её определения при этом данные могут выбираться в любом порядке. Имя хранилища должно идентифицировать его содержимое.

Внешняя сущность представляет собой сущность вне контекста системы, являющуюся источником или приемником данной системы. Предполагается, что объекты, представленные внешними сущностями не должны участвовать ни в какой обработке. Одна внешняя сущность может быть использована многократно на одной или нескольких диаграммах.

Для знакомства с процессом создания DFD-диаграмм, нужно открыть модель «Создание продукта», созданную на предыдущем занятии. Далее выбрать работу «Оформление заказа» и построить дочернюю диаграмму в нотации DFD (рис.Рисунок 2.21).

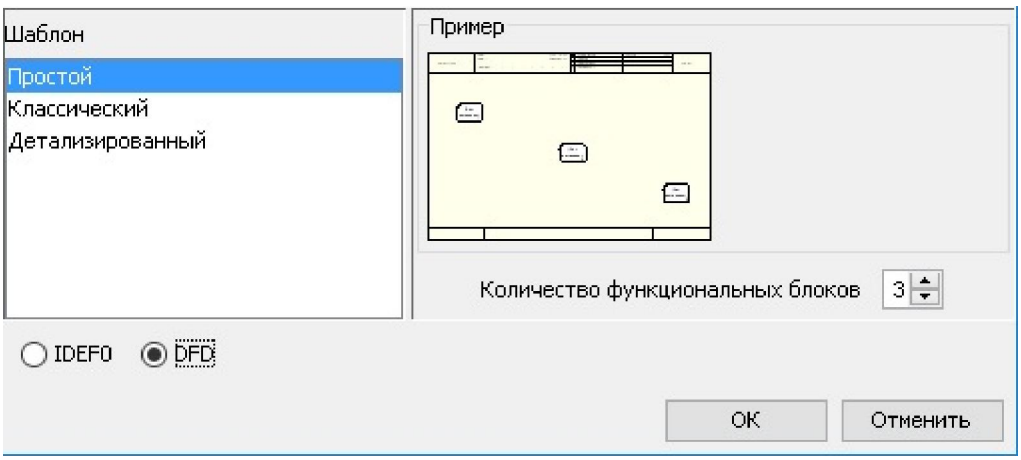


Рисунок 2.21 – Создание DFD-диаграммы

При формировании декомпозиции нужно внести на диаграмму имена работ: «Проверка и внесение клиента», «Заполнение формы заказа», «Формирование заявки на доставку».

Согласно нотации DFD диаграмма не должна иметь граничных стрелок-все стрелки должны начинаться и заканчиваться на работах, хранилищах данных или внешних сущностях. Поэтому, если строго следовать правилам нотации, следует: удалить все граничные стрелки на диаграмме DFD. И создать соответствующие внешние сущности и хранилища данных.

Для этого нужно добавить на диаграмму внешнюю сущность, выбрав соответствующую кнопку на панели инструментов, затем сделать по объекту двойной щелчок и в появившемся диалоговом окне выбрать «Задать DFD объект». Затем в появившемся диалоговом окне в контекстном меню выбрать пункт «Создать элемент». Название классификатора «Внешние сущности» можно ввести в созданную строку, дважды, медленно кликнув мышью по строке, или же нажав клавишу F2, предварительно

выделив нужную строку мышью. Затем нужно добавить к созданному классификатору элемент «Клиент» (рис. Рисунок 2.22). Аналогично создается внешняя сущность «Поставщики».

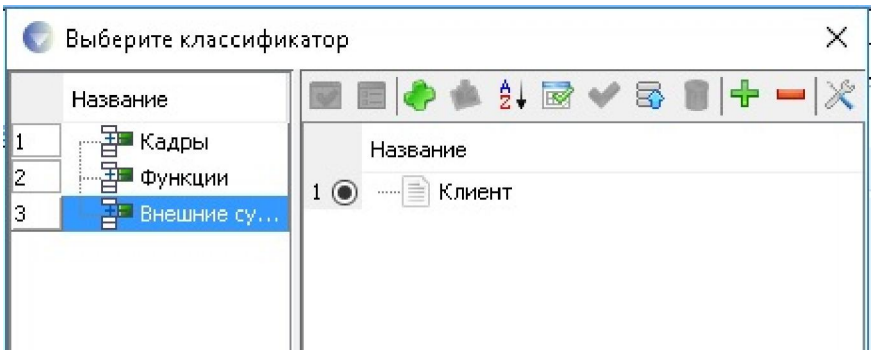


Рисунок 2.22 – Создание элемента Клиент классификатора Внешние сущности

Для внесения в модель хранилища используется кнопка . Затем нужно создать классификатор «Хранилища данных» и элемент для него «Заказы». Аналогично задаются элементы «Клиенты» и «Заявки на доставку». После этого нужно создать внутренние стрелки и именовать их (рис. Рисунок 2.23).После чего сохранить модель.

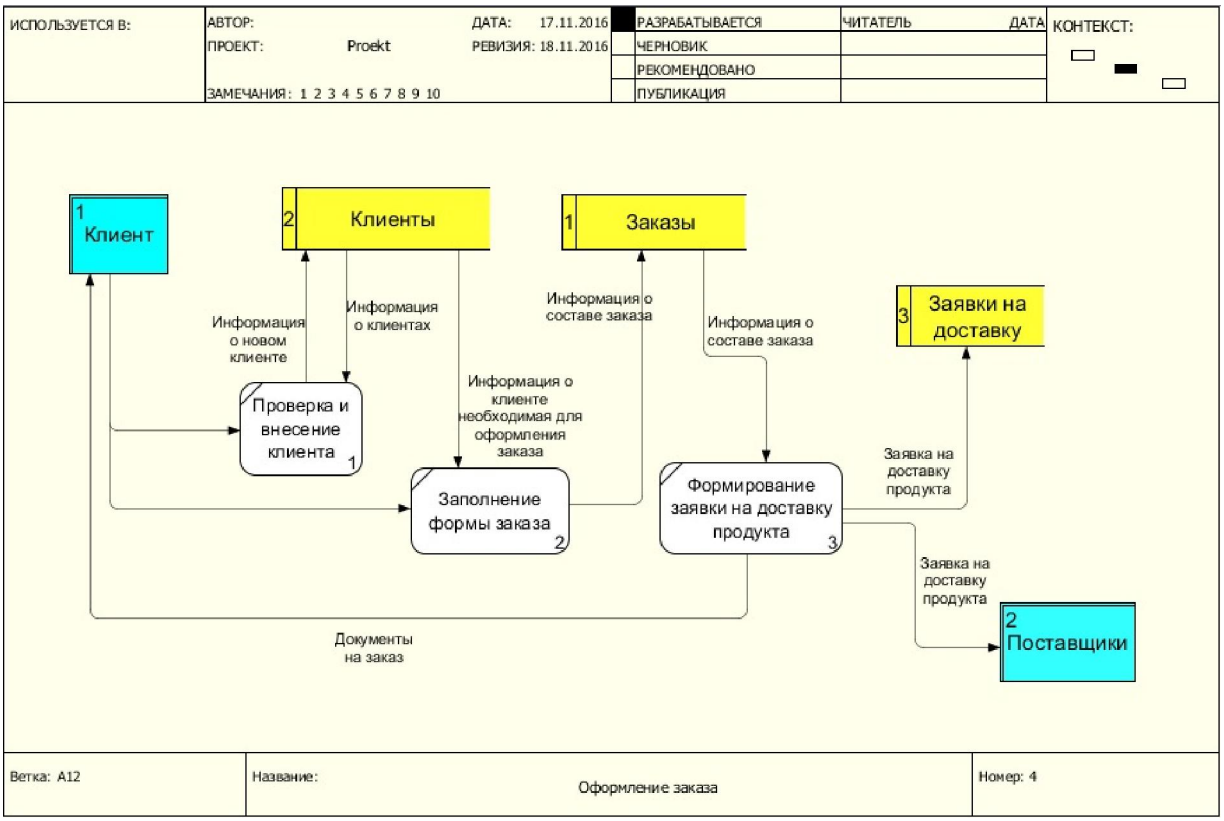


Рисунок 2.23 – DFD-диаграмма декомпозиции процесса оформления заказа

Самостоятельная работа

Открыть файл, в котором хранится модель, созданная по индивидуальному заданию. Требуется создать несколько DFD диаграмм для работ описывающих обработку информации, документооборот.

2.6 Учебная модель, описывающая деятельность компании

Рассмотрим деятельность вымышленной компании New Computer, которая существует 5 лет и занимается в основном сборкой и продажей настольных компьютеров

и ноутбуков. Годовой оборот компании составляет примерно 20 млн. руб. Компания закупает компоненты для компьютеров от трех независимых поставщиков, а не производит компоненты самостоятельно. Она только собирает и тестирует компьютеры. Компания реализует продукцию через магазины и специализируется на покупателях, для которых главный критерий при покупке – стоимость компьютера. Предполагаемый объем рынка для компании Computer в последующие 2 года – 50 млн. рублей. Несмотря на некоторое увеличение объема продаж, прибыли уменьшаются, растет конкуренция на рынке. Чтобы не потерять позиции, компания решает проанализировать текущие бизнес-процессы и реорганизовать их с целью увеличения эффективности производства и продаж. Основные бизнес-процессы в компании таковы:

15 продавцы принимают заказы клиентов;
16 операторы группируют заказы по типам компьютеров;
17 операторы собирают и тестируют компьютеры;
18 операторы упаковывают компьютеры согласно заказам;
19 кладовщик отгружает клиентам заказы.

В настоящее время компания Computer использует лицензионную бухгалтерскую информационную систему, которая позволяет оформить заказ, счет и отследить платежи по счетам. Улучшение деятельности компании должно касаться структуры управления компанией, эффективности производства и внутреннего контроля.

В результате реорганизация может потребовать внедрения новой корпоративной информационной системы (состоящей не только из одного бухгалтерского модуля). Однако перед тем как пытаться производить какие-то улучшения, необходимо разобраться в существующих бизнес-процессах.

На основании описательной части необходимо создать контекстную диаграмму. Цель - моделировать текущие (AS-IS) бизнес-процессы компании Computer. Точка зрения - директор. Основной функцией является - деятельность компании Computer. Стрелки контекстной диаграммы приведены в табл. 2.1.

Таблица 2.1 – Стрелки контекстной диаграммы

Наименование (Arrow Name)	Назначение (Arrow Definition)	Тип (Arrow Type)
Бухгалтерская система	Оформление счетов, оплата счетов, работа с заказами	Mechanism
Звонки клиентов	Запросы информации, заказы, тех.	Input
Правила и процедуры	Правила продаж, инструкции по сборке, процедуры тестирования, критерии	Control
Проданные продукты	Настольные и портативные компьютеры	Output

Необходимо декомпозировать контекстную диаграмму, разбив основную задачу на три. Описание работ приведено в табл. 2.2.

Таблица 2.2 – Описание работ

Наименование	Описание
Продажи и маркетинг	Телемаркетинг и презентации, выставки
Сборка и тестирование компьютеров	Сборка и тестирование настольных и портативных компьютеров
Отгрузка и получение	Отгрузка заказов клиентам и получение компонент от поставщиков

Результат показан на рис. 2.24.

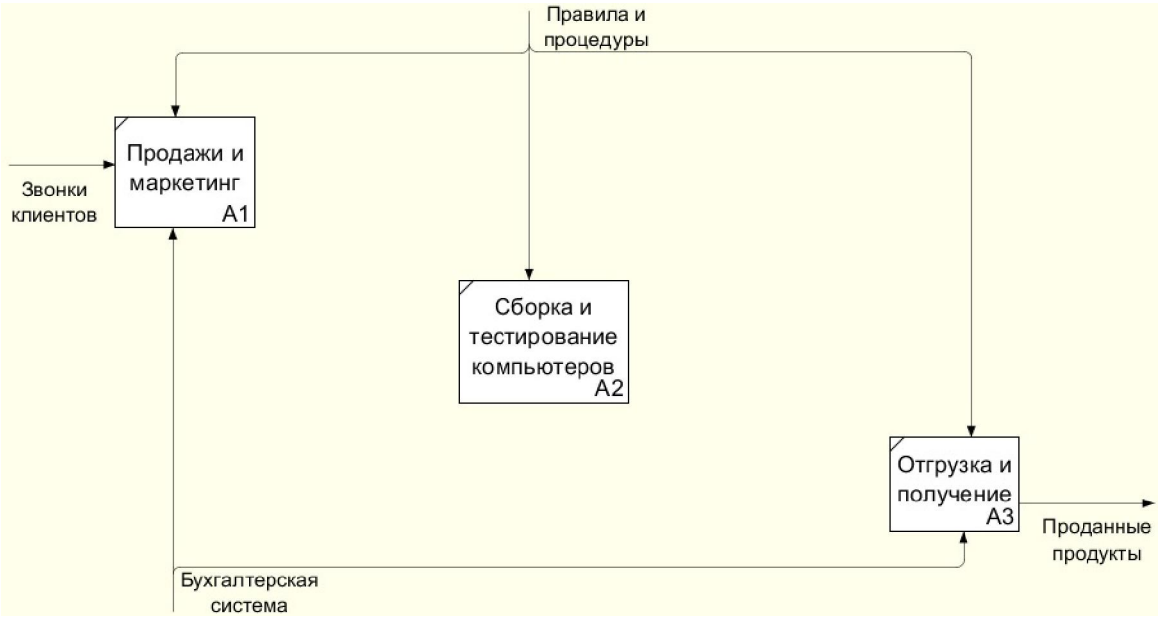


Рисунок 2.24 – Диаграмма декомпозиции

Далее необходимо изменить имена стрелок для большей ясности. Так, стрелку управления работы «Сборка и тестирование компьютеров» необходимо переименовать в «Правила сборки и тестирования» (описание: «Инструкции по сборке, процедуры тестирования, критерии производительности и т.д.»). Стрелку механизма работы «Продажи и маркетинг» переименовать в «Система оформления заказов».

Создать новые внутренние стрелки, как показано на рис. 2.25.

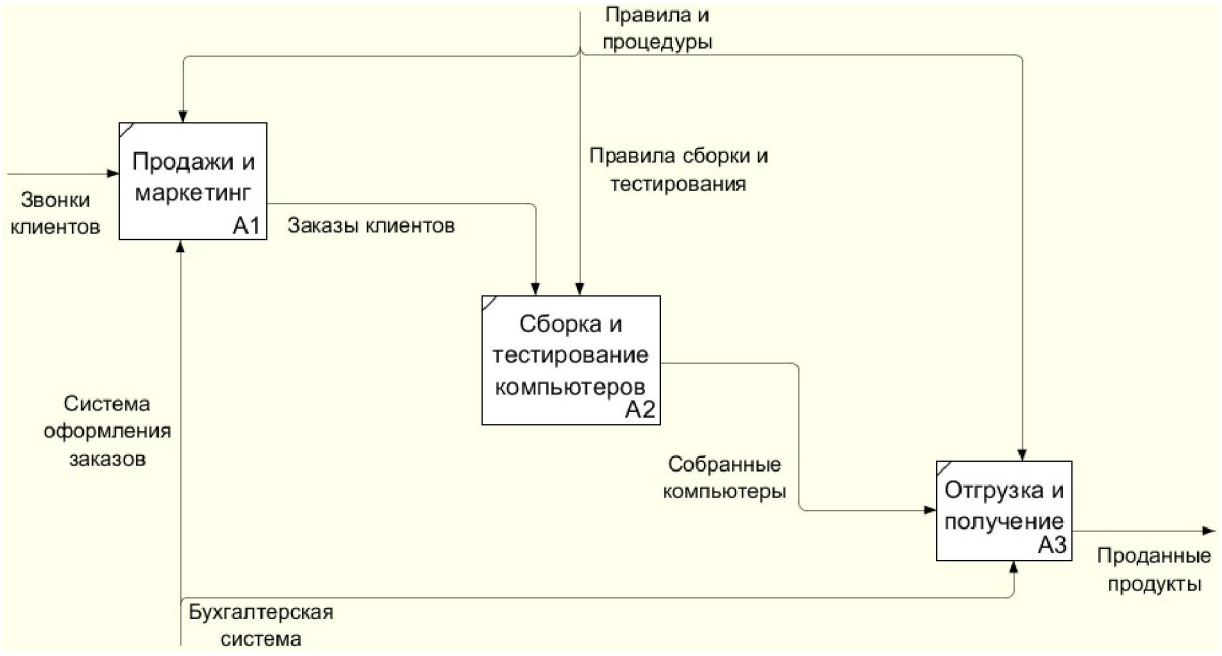


Рисунок 2.25 – Создание новых внутренних стрелок

Далее нужно создать стрелку обратной связи (по управлению) «Результаты сборки и тестирования», идущую от работы «Сборка и тестирование компьютеров» к работе «Продажи и маркетинг». Затем создать новую граничную стрелку выхода «Маркетинговые материалы», выходящую из работы «Продажи и маркетинг». Вид конечной диаграммы приведен на рис. 2.26.

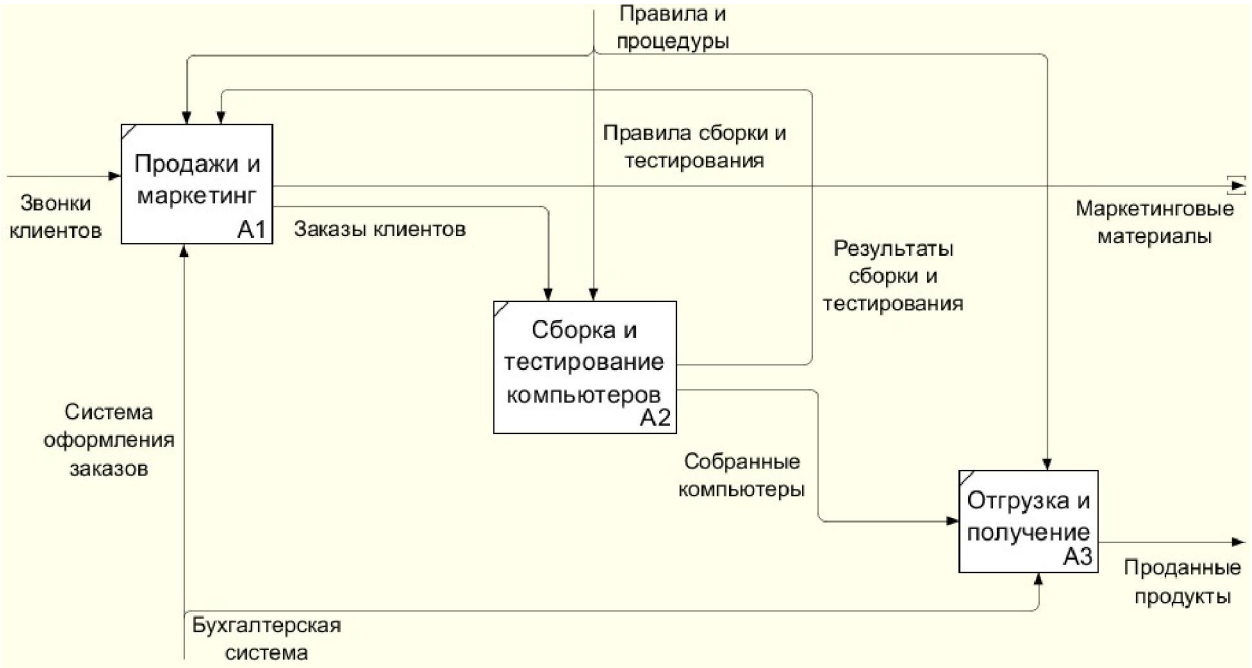


Рисунок 2.26 – Конечная дочерняя диаграмма первого уровня

Декомпозируем работу «Сборка и тестирование компьютеров» на основе следующей информации. Диспетчер координирует работу сборщиков, сортирует заказы, группирует их и дает указание на отгрузку компьютеров, когда они готовы. Каждые 2 часа Диспетчер группирует заказы – отдельно для настольных компьютеров и ноутбуков – и направляет на участок сборки.

Сотрудники участка сборки собирают компьютеры согласно спецификациям заказа и инструкций по сборке. Когда группа компьютеров, соответствующая группе заказов, собрана, она направляется на тестирование. Тестировщики тестируют каждый компьютер и в случае необходимости могут заменить неисправные компоненты. Тестировщики направляют результаты тестирования Диспетчеру, который на основании этой информации принимает решение о передаче компьютеров, соответствующих группе заказов на отгрузку (табл. 2.3). Вид полученной диаграммы приведен на рис. 2.27.

Таблица 2.3 – Описание процессов для работы «Сборка и тестирование компьютеров»

Наименование	Описание
Отслеживание расписания и управление сборкой и тестированием	Просмотр заказов, установка расписания выполнения заказов, просмотр результатов тестирования, формирование групп заказов на
Сборка настольных компьютеров	Сборка настольных компьютеров в соответствии с инструкциями и указаниями диспетчера
Сборка ноутбуков	Сборка ноутбуков в соответствии с инструкциями и указаниями диспетчера
Тестирование компьютеров	Тестирование компьютеров и компонент. Замена неработающих компонент

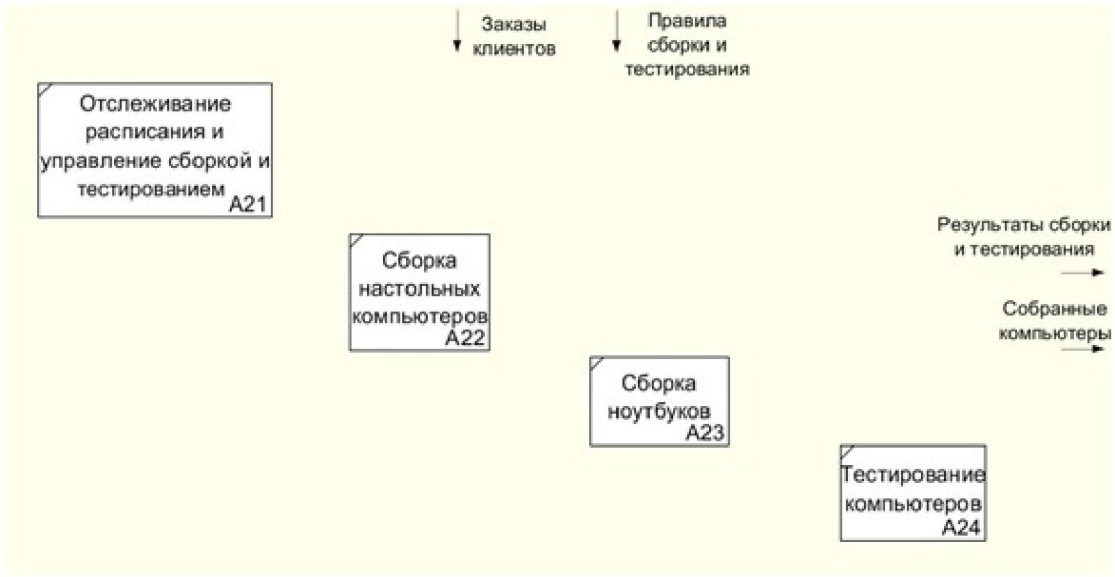


Рисунок 2.27 – Блоки сборки и тестирования

Требуется задать на диаграмме двадцать одну стрелку, их имена и описания в соответствие с табл. 2.4.

Таблица 2.4 – Описание стрелок для декомпозиции работы «Сборка и тестирование компьютеров»

№	Стрелка	Источник	Тип источника	Назначение	Тип назначения
1	Заказ клиентов	Граница (стрелка)	Управление	Отслеживание расписания и управление сборкой и тестированием	Управление
2	Правила сборки и тестирования	Граница (стрелка)	Управление	Сборка настольных компьютеров	Управление
3	Результаты сборки и тестирования	Сборка настольных компьютеров	Выход	Граница (стрелка)	Выход
4	Собранные компьютеры	Тестирование компьютеров	Выход	Граница (стрелка)	Выход
5	Правила сборки и тестирования	Стрелка от границы	Управление	Сборка ноутбуков	Управление
6	Правила сборки и тестирования	Стрелка от границы	Управление	Тестирование компьютеров	Управление
7	Результаты сборки и тестирования	Сборка ноутбуков	Выход	Стрелка к границе	Выход
8	Результаты сборки и тестирования	Тестирование компьютеров	Выход	Стрелка к границе	Выход
9	Заказы на настольные компьютеры	Отслеживание расписания и управление	Выход	Сборка настольных компьютеров	Управление

		сборкой и тестированием			
10	Заказы на ноутбуки	Отслеживание расписания и управление сборкой и тестированием	Выход	Сборка ноутбуков	Управление
11	Настольные компьютеры	Сборка настольных компьютеров	Выход	Тестирование компьютеров	Вход
12	Ноутбуки	Сборка ноутбуков	Выход	Тестирование компьютеров	Вход
13	Результаты тестирования	Тестирование компьютеров	Выход	Отслеживание расписания и управление сборкой и тестированием	Вход
14	Указание передать компьютеры на отгрузку	Отслеживание расписания и управление сборкой и тестированием	Выход	Тестирование компьютеров	Управление
15	Персонал производственного отдела	Граница (туннель)	Механизм	Сборка настольных компьютеров	Механизм
16	Персонал производственного отдела	Граница (стрелка)	Механизм	Сборка ноутбуков	Механизм
17	Диспетчер	Персонал производственного отдела	Механизм	Отслеживание расписания и управление сборкой и тестированием	Механизм
18	Тестировщик	Персонал производственного отдела	Механизм	Тестирование компьютеров	Механизм
19	Компоненты	Граница (туннель)	Вход	Сборка настольных компьютеров	Вход
20	Компоненты	Граница (стрелка)	Вход	Сборка ноутбуков	Вход
21	Компоненты	Граница (стрелка)	Вход	Тестирование компьютеров	Вход

Окончательный вид диаграммы приведен на рис. 2.28.

Далее необходимо декомпозировать работу «Продажи и маркетинг». Работа по продажам и маркетингу заключается в ответах на телефонные звонки клиентов, предоставлении клиентам информации о ценах, оформлении заказов, внесении заказов в информационную систему и исследовании рынка. Итоговая диаграмма приведена на рис. 2.29.

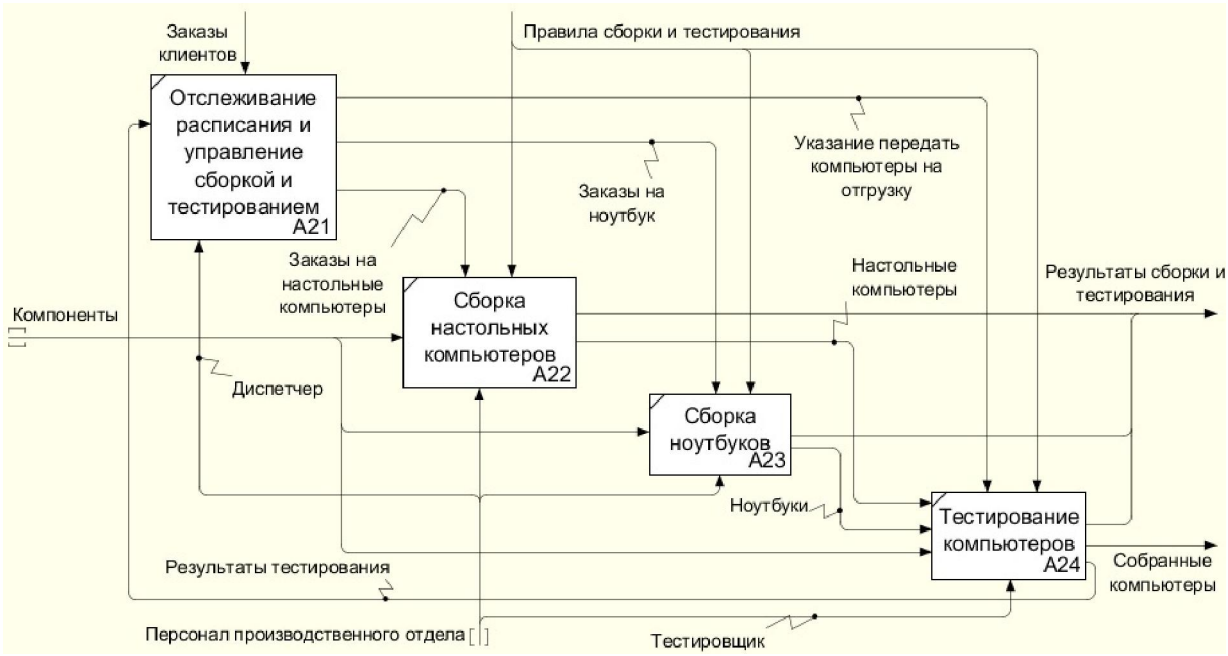


Рисунок 2.28 – Диаграмма декомпозиции работы Сборка и тестирование компьютеров

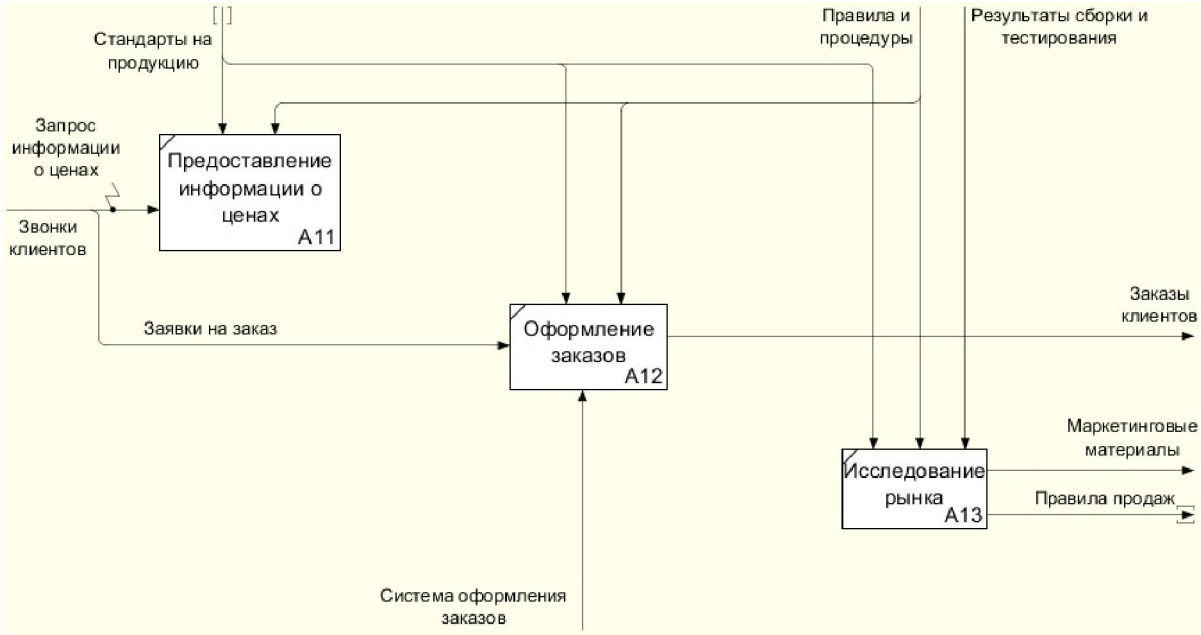


Рисунок 2.29 – Диаграмма декомпозиции работы Продажи и маркетинг

3. Объектно-ориентированное проектирование: UML

3.1 Знакомство с пакетом StarUML

Унифицированный язык моделирования (Unified Modeling Language, UML) является графическим языком для визуализации, специфицирования, конструирования и документирования систем, в которых большая роль принадлежит программному обеспечению. С помощью UML можно детально описать систему, начиная разработку с концептуальной модели с ее бизнес-функциями и процессами, а также описать особенности реализации системы, такие как классы программного обеспечения системы, схему базы данных. UML также позволяет разрабатывать сложные системы быстро и качественно.

Как язык графического визуального моделирования UML имеет свою нотацию – принятые обозначения. Нотация обеспечивает семантику языка, является способом унификации обозначений визуального моделирования, обеспечивает всестороннее представление системы, которое сравнительно легко и свободно воспринимается человеком. Моделирование с помощью UML осуществляется поэтапным построением ряда диаграмм, каждая из которых отражает какую-то часть или сторону системы либо ее замысла.

Напомним, что диаграмма – это графическое представление множества элементов. Диаграммы, как правило, изображаются в виде графов с вершинами (сущностями) и ребрами (отношениями). Диаграммы подчиняются нотации UML и изображаются в соответствии с ней.

Основные диаграммы UML:

- 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).

Построения этих диаграмм достаточно для полного моделирования системы. В данном пособии рассматриваются основные элементы нотации диаграмм и принципы их построения с помощью инструментального средства StarUML.

Основная структурная единица в StarUML – это проект. Проект сохраняется в одном файле в формате XML с расширением «.UML». Проект может содержать одну или несколько моделей и различные представления этих моделей (View) – визуальные выражения информации, содержащейся в моделях. Каждое представление модели содержит диаграммы – визуальные образы, отображающие определенные аспекты модели.

Новый проект будет автоматически создан при запуске программы StarUML. При этом будет предложено в диалоговом окне выбрать один из подходов (Approaches), поддерживаемых StarUML, рис. 3.1. Существуют различные методологии моделирования информационных систем, компании-разработчики систем также могут разрабатывать свои методологии. Следовательно, на начальной стадии проектирования необходимо

определить основные положения методологии или выбрать одну из уже существующих. Для того чтобы согласовать между собой различные элементы и этапы моделирования, StarUML предлагает концепцию подходов. В данном пособии используется Rational Appoach. После того как выбран один из предложенных подходов, появится основное окно программы, рис. 3.2.

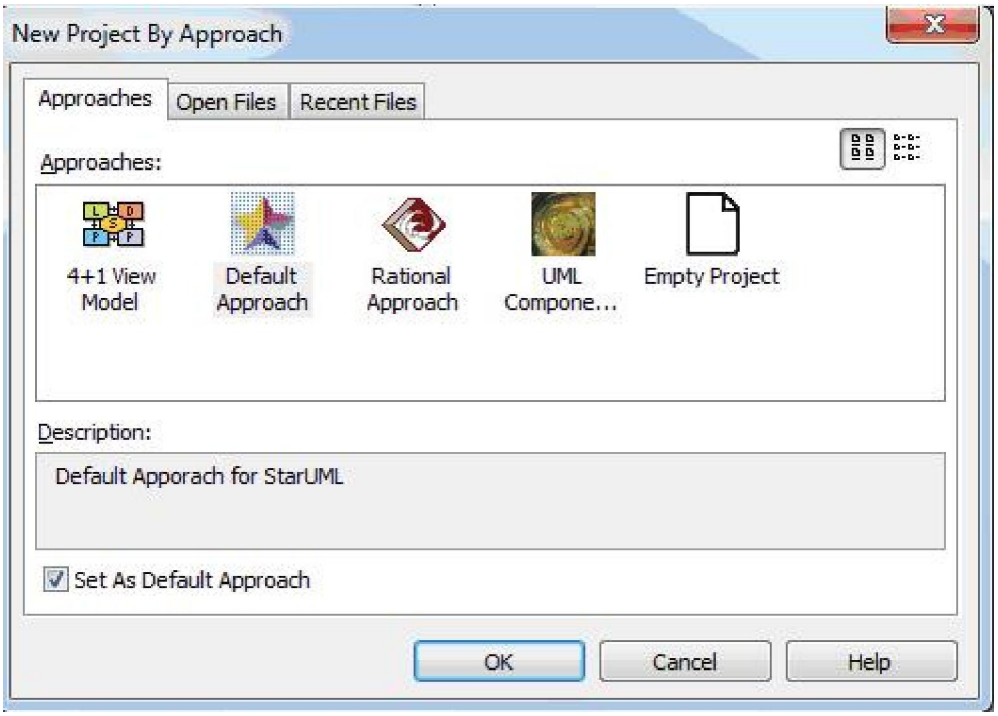


Рисунок 3.1 – Окно создания проекта и выбора подхода

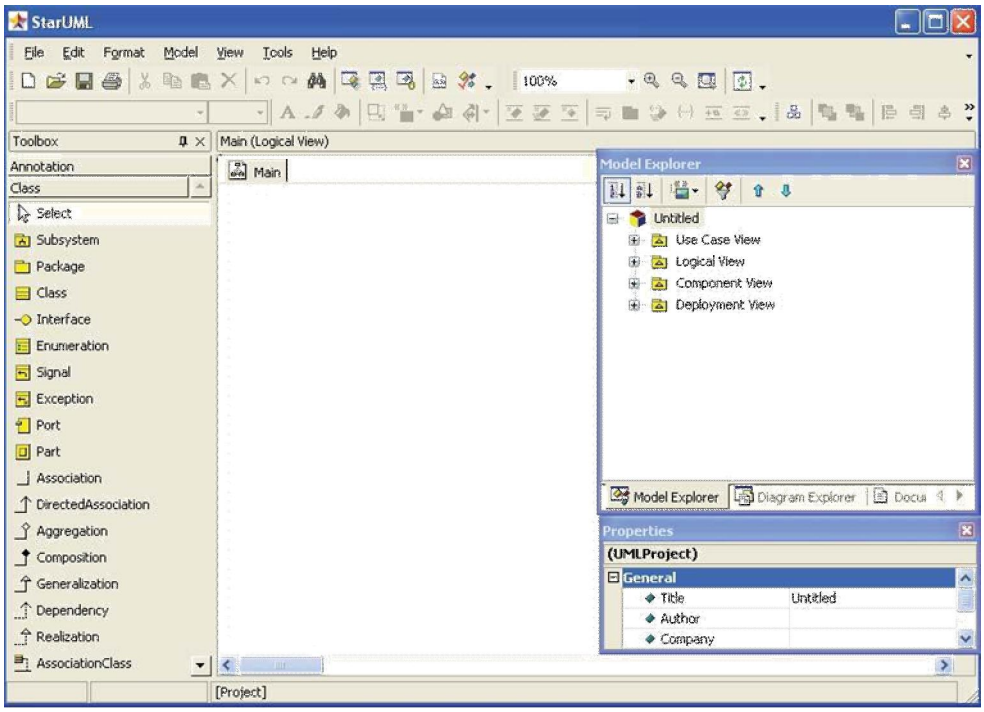


Рисунок 3.2 – Основное окно программы StarUML

Дадим некоторые пояснения.

Use Case View – представление требований к системе, описывает, что система должна делать;

Logical View – логическое представление системы, описывает, как система должна быть построена;

Component View – представление реализации, описывает зависимость между программными компонентами;

Deployment View – представление развертывания, описывает аппаратные элементы, устройства и программные компоненты.

В верхней части окна расположено главное меню, кнопки быстрого доступа. Слева расположена панель элементов (Toolbox) 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 находится инспектор модели, на котором можно найти вкладки навигатора модели Model Exploer, навигатора диаграмм Diagram Exploer, окно редактора свойств Properties, окно документирования элементов модели Documentation и редактор вложений Attachments.

Управлять видом инспектора модели, панели элементов, закрывать и открывать редакторы инспектора можно с помощью пункта меню View (рис. 3.3). Если рядом с пунктом меню стоит «галочка», то элемент активен и его можно видеть в окне программы или открыть на доступных вкладках инспектора модели.

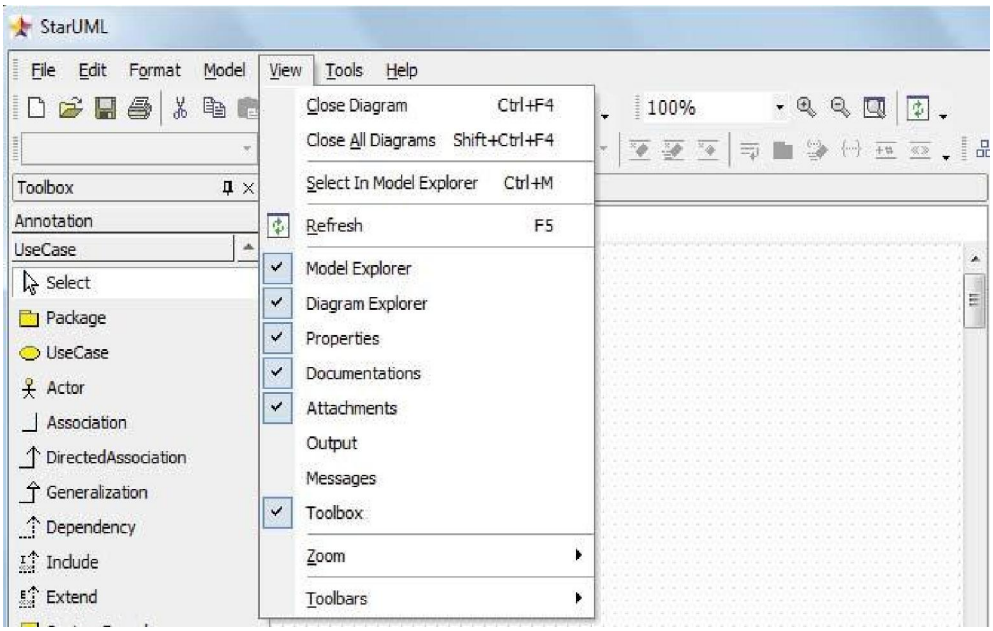


Рисунок 3.3 – Управление видом инспектора модели с помощью меню View

Иерархическая структура проекта отображается справа на навигаторе модели (Model Exploer). В зависимости от выбранного подхода на навигаторе модели будут отображены различные пакеты представлений модели. Каждый пакет представления будет содержать элементы моделей и диаграмм, которые создает пользователь. В самом начале каждое представление содержит одну диаграмму с именем Main. Если щелкнуть по ней два раза, то откроется рабочее поле этой диаграммы и соответствующая панель инструментов. Так, если щелкнуть два раза по диаграмме Main представления Use Case View, то откроется рабочее этой диаграммы и ее панель элементов, рис. 3.4.

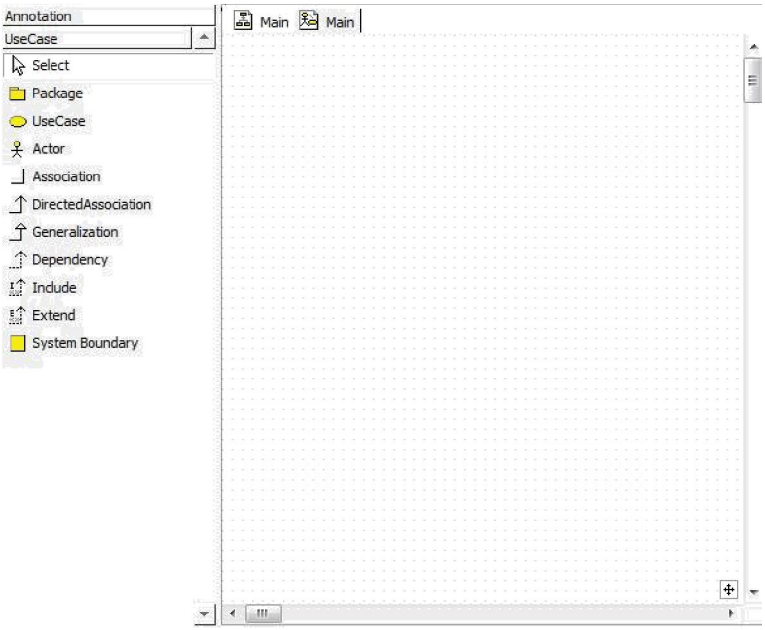


Рисунок 3.4 – Рабочее поле диаграммы прецедентов Main и ее панель элементов

Ниже иерархии представлений отображаются свойства выделенного элемента модели или диаграммы (в данном случае свойства диаграммы Main, так как она выделена в навигаторе модели), рис. 3.5.

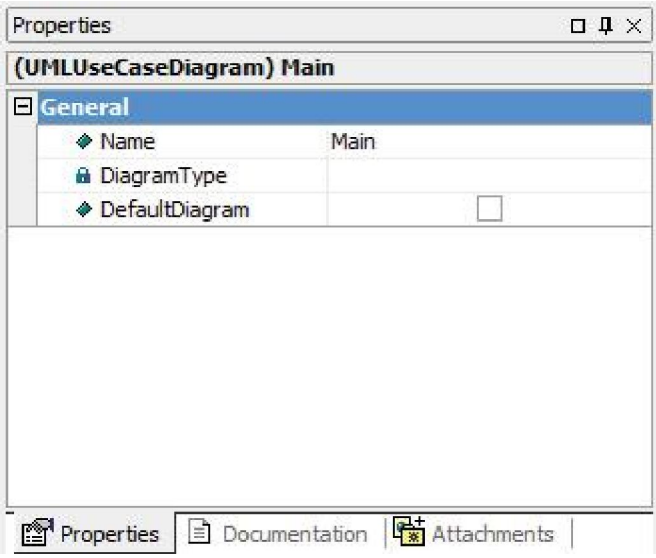


Рисунок 3.5 – Редактор свойств

Для получения практических навыком рассмотрим особенности проектирования системы заказов интернет-магазина «Style».

Постановка задачи. Магазин занимается продажей детской и взрослой одежды и обуви различных брендов. Покупатель просматривает каталог и делает заказ. Предполагаем, что потенциальный клиент заходит на сайт магазина, он может нажать кнопку просмотра (или загрузки) каталога, далее может положить понравившийся товар в корзину, изменить корзину и, приняв решение о покупке товаров, перейти из корзины к оформлению заказа. Для того чтобы корректно создать систему, отвечающую всем требованиям заказчика, необходимо четко представить себе ее основные бизнес-функции и выяснить предъявляемые к системе требования. Для этого необходимо провести обследование компании и построить ее полную бизнес-модель. Поскольку

рассматриваемый пример является учебным, выполнить такое обследование не представляется возможным, поэтому воспользуется следующим описанием системы.

Каждый товар в каталоге описывается артикулом, размерным рядом, ценой и фото с кратким описанием. Покупатель может загрузить каталог товаров. Каталог не содержит разделы, имеет блочную структуру, состоит из набора товаров с фото, ценой размерами. Покупатель складывает понравившиеся товары в корзину, при этом выбирая размер и количество необходимого товара данного артикула. Корзину можно изменить: просмотреть, удалить товар, изменить количество позиций одного артикула, вернуться в каталог. Когда покупатель делает заказ, он вводит свои личные данные, телефон оплачивает его по банковской карте (если заказ не оплачен, то он и не сделан). После того как сделан заказ, его можно забрать со склада через 1 рабочий день. Данные о заказе поступают сотруднику магазина, назовем его сотрудником отдела продаж, он проверяет наличие товаров и передает его кладовщику на комплектацию. Кладовщик, собрав заказ, делает отметку о готовности. Заказ выдается со склада кладовщиком. Кладовщик выдает заказ и отмечает в системе, что заказ выдан. Магазин не занимается доставкой заказов, не делает скидок. Для того чтобы ограничить масштаб задачи, в рассматриваемом случае не учтена система снабжения магазина новыми товарами. Этим занимается другая система «Склад». Информация о проданных товарах (т.е. сделанных заказах) поступает также в систему «Склад». Поскольку на протяжении от создания до выдачи заказа, он проходит разные стадии, то будет разумно ввести понятие статуса заказа. Сотрудники магазина могут статус заказа изменять, а покупатель может проследить за сборкой заказа. В таком случае рассматриваемая система предоставляет еще одну функцию: узнать статус заказа.

Поведение системы (т.е. функциональность, которую она обеспечивает) описывают с помощью функциональной модели, которая отображает системные прецеденты (use cases, случаи использования), системное окружение (действующих лиц, актеров, actors) и связи между ними (use cases diagrams).

Диаграмма вариантов использования (диаграмма прецедентов, use case diagram) – это диаграмма, на которой изображаются отношения между актерами и вариантами использования. С помощью этой диаграммы можно:

- Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
- Сформулировать общие требования к функциональному поведению проектируемой системы.
- Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- 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.

Вариант использования (прецедент, use case) – описание множества последовательности действий (включая варианты), выполняемых системой для того, чтобы актер мог получить определенный результат.

Каждый вариант использования должен быть независимым в том смысле, что если он всегда выполняется совместно с другим вариантом использования, то, скорее всего, это один прецедент, а не два, либо они связаны отношением включения или расширения, о чем речь пойдет позже. Как следует из определения, прецедент (или вариант использования) должен обладать результирующей ценностью для актера, актер должен получить некоторый результат исполнения прецедента. Скорее всего, после исполнения прецедента в системе произойдут некоторые изменения: появятся новые данные, изменится поведение. Каждый вариант использования должен исполняться от начала до конца. 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. 3.6.

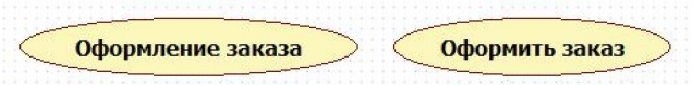


Рисунок 3.6 – Варианты использования (прецеденты)

Для того чтобы создать прецедент, щелкните по овальному символу прецедента на панели элементов слева от рабочего поля диаграммы, а затем щелкните по тому месту на рабочем поле диаграммы, в которое необходимо поместить прецедент. Аналогичным образом создается актер. Когда элемент помещается на поле диаграммы, он становится доступен для редактирования имени и некоторых свойств. В выделенное поле введите новое имя прецедента или актера (рис. 3.7).

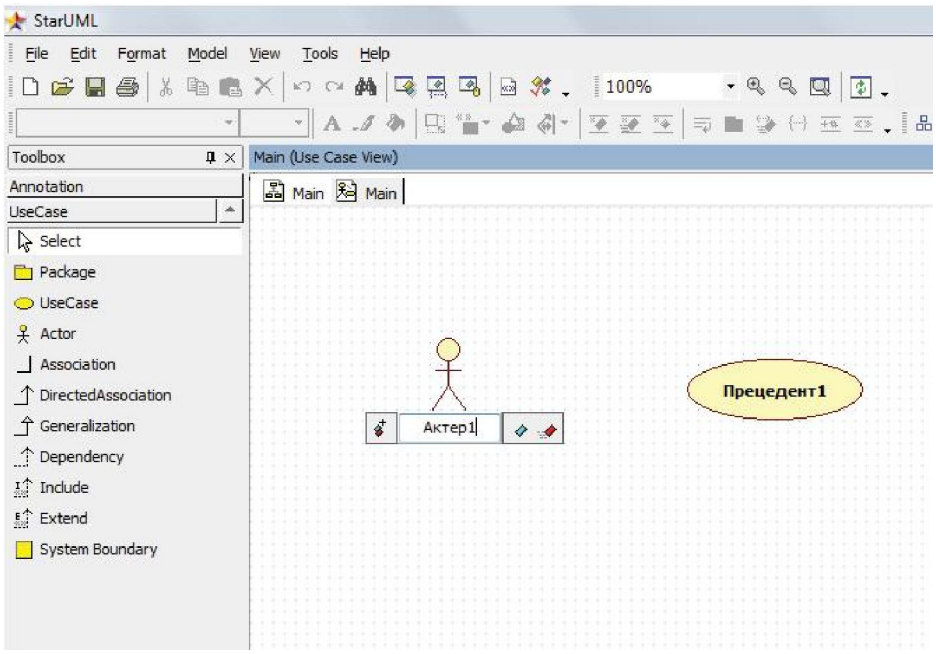


Рисунок 3.7 – Именование элементов диаграммы

Одним из наиболее важных (и дорогостоящих) этапов проектирования информационных систем является этап определения требований к системе. Если

требования заказчика информационной системы разработчиками будут определены не корректно, то в итоге заказчик может получить совсем не ту систему, которую он ожидал. Моделирование прецедентов и актеров помогает нам лучше понять требования, предъявляемые к системе, и согласовать их с заказчиком с помощью демонстрации и обсуждения диаграммы прецедентов. Прецеденты и актеры – это отражение требований к системе, они показывают, кто и для чего будет использовать будущую систему.

Определим актеров и прецеденты системы заказов магазина «Style». Напомним, что покупатель делает заказ, складывая товары в корзину. Возможна только одна форма оплаты: банковской картой по интернету, невозможно оформление заказа без оплаты. Заказ имеет статус: оплачен, передан на комплектацию, собран, получен. Статус заказа изменяется автоматически либо сотрудником магазина. Покупатель может узнать статус своего заказа по уникальному номеру заказа.

Поскольку разрабатываемая система не занимается поставками товаров в магазин (этим занимается другая система «Склад»), то с проектируемой системой взаимодействуют покупатель, сотрудники магазина и система «Склад». С системой взаимодействуют сотрудник отдела продаж, который проверяет оплату заказа и отправляет его на комплектацию, и кладовщик, который собирает заказ и выдает его покупателю. С точки зрения бизнеса – это две разных должности, выполняющих разные функции, но с точки зрения системы они играют одну роль сотрудника, изменяющего статус заказа покупателя с использованием программного обеспечения моделируемой системы. В этом смысле для системы нет разницы между сотрудником отдела продаж и кладовщиком. Выбирая действующих лиц, нужно помнить о том, что мы должны отразить их роль, а не должность. Введем обобщающее сотрудников действующее лицо – Сотрудник. Другой пример: сотрудник магазина «Style» (например, кладовщик) может выступать в роли сотрудника и общаться с системой как сотрудник магазина, а может выступать и в роли покупателя, сделав заказ в магазине. Несмотря на то, что физически это один человек, он выступает в роли двух актеров: покупателя и сотрудника. Итак, актеры системы заказов магазина «Style» будут: Покупатель, Сотрудник, система «Склад».

Покупатель использует проектируемую систему для того, чтобы заказать вещи, он просматривает каталог, добавляет понравившиеся ему товары в корзину, открывает корзину, удаляет из нее товары или изменяет их количество и, наконец, может оформить свой заказ, при этом его оплатив. В конечном итоге результат использования системы покупателем будет получен, если он выполнил все эти действия от начала до конца. Поэтому не будем разделять заказ товаров на несколько прецедентов, а выделим только один: Заказ товаров. Покупатель, сделав заказ в магазине «Style», может в дальнейшем узнавать статус своего заказа, это тоже случай использования системы, назовем его Получение информации о заказе.

Сотрудник должен изменять статус сделанного заказа, для него определим прецедент Управление статусом заказа.

Система «Склад» должна получать информацию о сделанных заказах для возможности управления наличием товаров на складе, для нее также должен быть доступен прецедент Получение информации о заказе.

Итак, прецеденты системы заказов магазина «Style»: Заказ товаров, Управление статусом заказа, Получение информации о заказе.

Связи и взаимоотношения, существующие между элементами модели, в UML описываются с помощью отношений, изображаемых на диаграммах.

Отношение (relationship) – это семантическая связь между отдельными элементами модели. Между актерами и прецедентами диаграммы вариантов использования могут существовать разного рода отношения, показывающие, что экземпляры действующих лиц и вариантов использования взаимодействуют друг с другом. Действующие лица могут взаимодействовать с несколькими прецедентами и между собой, равно как и прецеденты могут быть связаны между собой особого типа отношениями. В основном на диаграммах прецедентов используются следующие типы отношений:

- ассоциации (association relationship);
- включения (include relationship);
- расширения (extend relationship);
- обобщения (generalization relationship).

Ассоциация – это структурное отношение, показывающее, что объект неким образом связан с другим объектом. Отношение этого типа используется не только на диаграммах прецедентов, но и на других диаграммах. Если между элементами модели показано отношение ассоциации, то это означает, что между ними существует семантическая связь. Ассоциативное отношение может быть направленным. В этом случае направление связи показывает, кто является инициатором. Если отношение направленно от актера к прецеденту, то это означает, что актер инициирует выполнение прецедента.

На рис. 3.8 показано, что Покупатель в системе заказов магазина «Style» инициирует выполнение прецедента Заказ товаров.

Рисунок 3.8 – Отношение ассоциации между актером и прецедентом

Для создания отношения между элементами диаграммы щелкнув, по изображению соответствующего отношения на панели элементов справа, провести линию от одного элемента к другому, удерживая левую кнопку мыши, рис. 3.9.

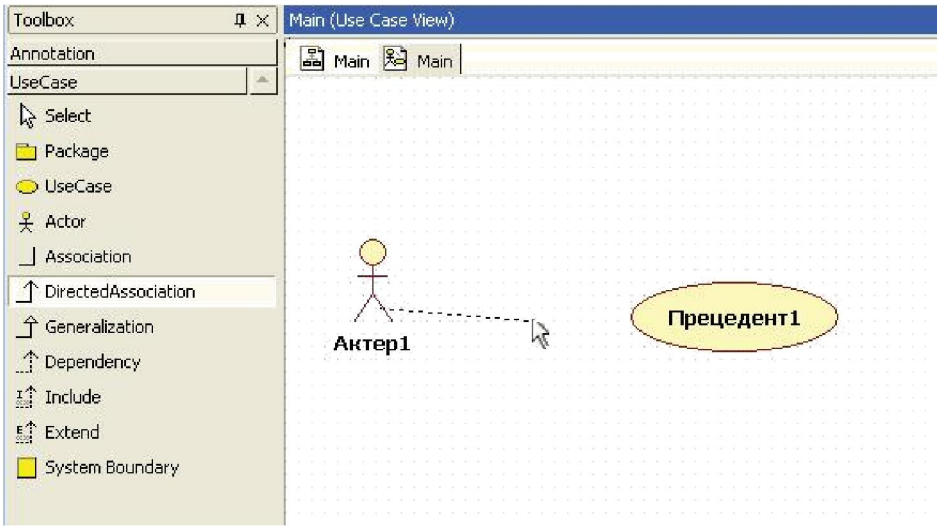


Рисунок 3.9 – Создание отношений между элементами

Чтобы удалить элемент с диаграммы достаточно щелкнуть левой кнопкой мыши по этому элементу, а затем нажать кнопку Delete, либо щелкнуть правой кнопкой мыши по элементу и в контекстном меню выбрать Edit -> Delete, рис. 3.10.

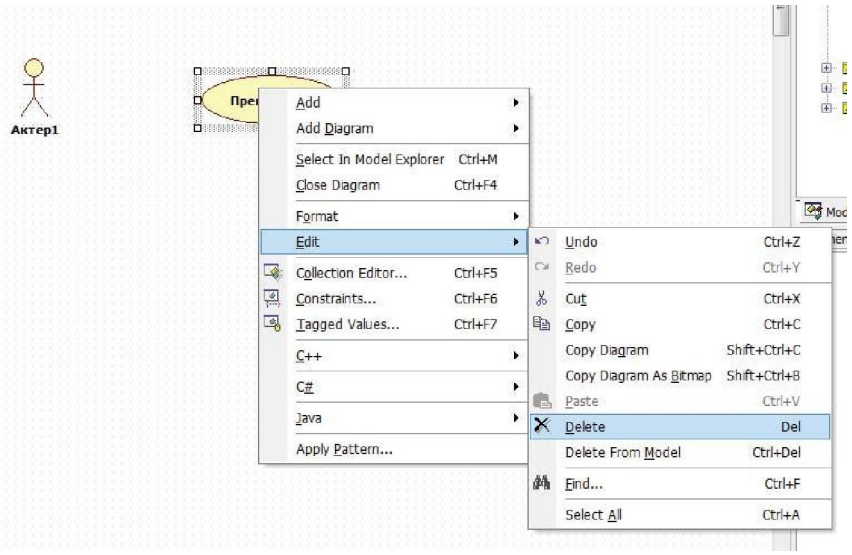


Рисунок 3.10 – Удаление элемента диаграммы

Стоит иметь в виду, что элемент был удален с диаграммы, но не из модели, рис. 3.11. Его можно найти в навигаторе модели, несмотря на то, что на диаграмме он больше не отображается (элемент Прецедент1):

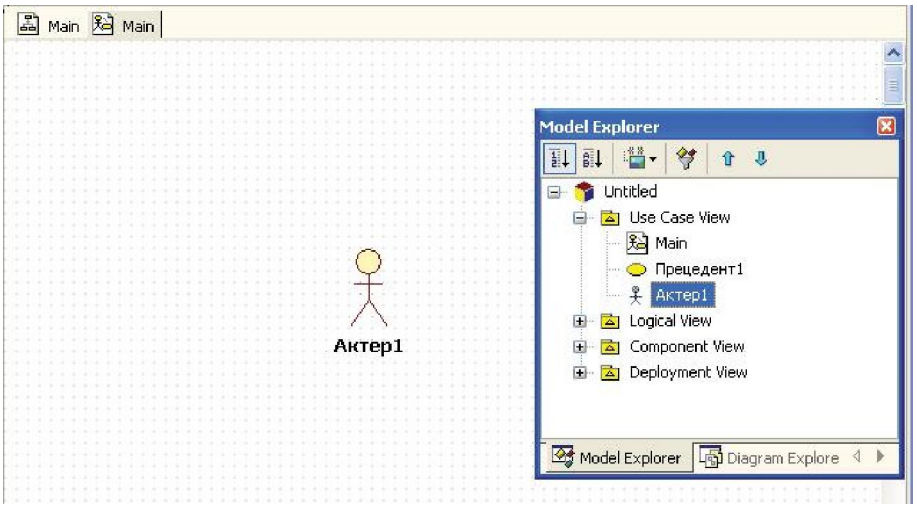


Рисунок 3.11 – Элемент Прецедент1 удален с поля диаграммы, но отображается в навигаторе модели

Если необходимо вернуть удаленный элемент на диаграмму, то это можно сделать, перетащив его с навигатора модели на поле диаграммы. Для того чтобы удалить элемент из модели нужно щелкнуть по нему на диаграмме или по его изображению в навигаторе модели правой кнопкой мыши и в контекстном меню выбрать пункт Delete from Model. Элемент будет полностью удален. Описанные выше способы добавления и удаления элементов и отношений могут быть использованы для построения диаграмм любых типов.

Включение (include) говорит о том, что исходный прецедент явным образом включает в себя поведение целевого. Другими словами, включение создается, когда один прецедент использует другой. При этом исполнение базового прецедента невозможно без исполнения используемого. Изображается включение в виде пунктирной стрелки с надписью <<include>>, которая направлена от базового элемента к используемому.

В системе заказов магазина «Style» невозможен заказ товаров без оплаты. На диаграмме прецедентов это можно отразить так, как это показано на рис. 3.12.

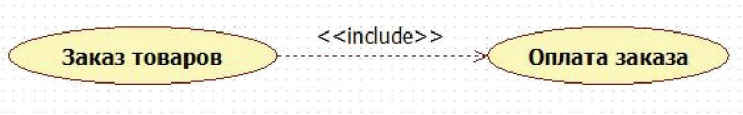


Рисунок 3.12 – Отношение включения между прецедентами

Расширение (extend) показывает, что целевой прецедент расширяет поведение исходного. Используемый прецедент выполняется не всегда вместе с базовым, а только при выполнении дополнительных условий, таким образом, расширяя функциональность базового элемента. Изображается расширение пунктирной стрелкой с надписью <<extend>>, направленной от используемого варианта использования к базовому.

При заказе товаров в системе заказов магазина «Style» покупатель может изменить содержание корзины перед тем, как оформить заказ окончательно, а может оставить корзину без изменений. Изменение корзины – это опция, которую на диаграмме вариантов использования можно изобразить с помощью расширения, рис. 3.13.

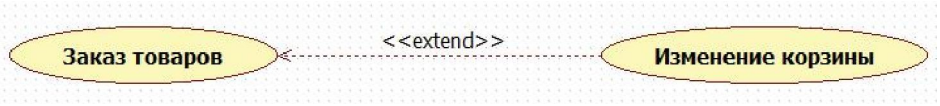


Рисунок 3.13 – Отношение расширения между прецедентами

Обозначения отношений <<include>> и <<extend>> есть не что иное, как обозначения стереотипов, которые широко используются в UML для создания новых элементов модели путем расширения функциональности базовых элементов.

Стереотип (Stereotype) – это механизм, позволяющий категоризировать элементы модели. С помощью стереотипов мы можем создавать своего рода подтипы типов. Это позволяет UML иметь минимальный набор элементов, которые могут быть дополнены при необходимости для создания связующих базовых элементов в системе. В UML стереотип обозначается именем, которое записывается в двойных угловых скобках: <<имя стереотипа>>. Пользователь может создавать собственные стереотипы на основе уже имеющихся типов. Также существуют и стандартные, заранее определенные стереотипы нотации UML. Так, отношение зависимости расширяется для прецедентов и актеров с помощью двух стереотипов <<include>> и <<extend>>.

Два и более актера могут иметь общие свойства, т.е. взаимодействовать с одним и тем же множеством вариантов использования одинаковым образом. Такая общность свойств и поведения представляется в виде отношения обобщения с другим, возможно, абстрактным актером, который моделирует соответствующую общность ролей.

Обобщение (Generalization) – это отношение между общей сущностью и ее конкретным воплощением. На диаграммах обобщение обозначается стрелкой с не закрашенным треугольником на конце, направленной от частного элемента к общему.

Для изменения статуса заказов в магазине «Style» с проектируемой системой будут работать сотрудник отдела продаж и кладовщик. На диаграмме можно показать с помощью отношения обобщения взаимосвязь между актером Сотрудник и актерами Сотрудник отдела продаж и Кладовщик, рис. 3.14.

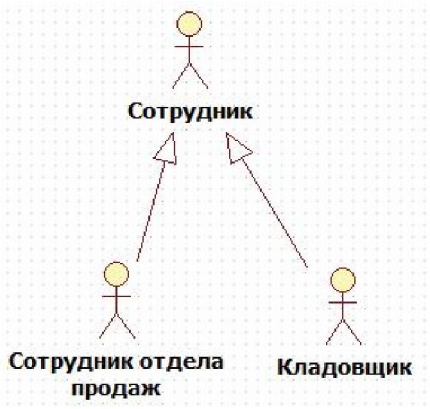


Рисунок 3.14 – Отношение обобщения между актерами

Актеры, прецеденты и отношения – это основные элементы нотации диаграмм вариантов использования. Диаграмма вариантов использования помогает отобразить основные требования к моделируемой системе и обеспечить взаимопонимание функциональности системы между разработчиком и заказчиком. Можно построить одну, главную диаграмму прецедентов, на которой будут отражены границы системы (актеры) и ее основная функциональность (прецеденты). Для более подробного представления системы допускается построение вспомогательных диаграмм прецедентов.

Для системы заказов магазина «Style» на данный момент определены: Покупатель, Сотрудник, Система «Склад» и прецеденты Заказ товаров, Управление статусом заказа, Получение информации о заказе. Вид основной диаграммы прецедентов приведен на рис. 3.15.

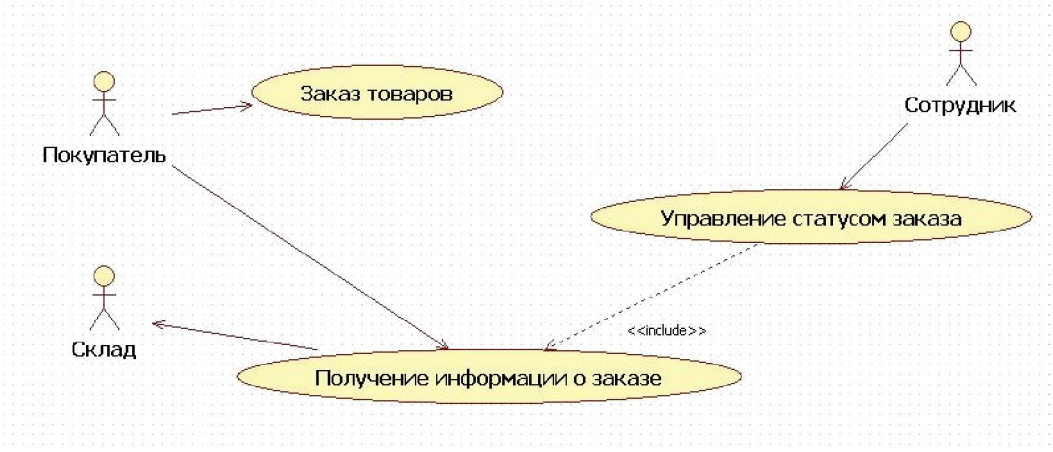


Рисунок 3.15 – Основная диаграмма вариантов использования системы заказов магазина «Style»

Для актера Покупатель и прецедента Заказ товаров установили отношение направленной ассоциации: Заказ товаров инициализируется Покупателем. Сотрудник имеет возможность управлять статусом заказа, при этом он непременно участвует в прецеденте Получение информации о заказе. Направленную ассоциацию от Получение информации о заказе к актеру Система «Склад» можно понимать как автоматическую передачу данных из моделируемой системы в систему снабжения товарами «Склад».

В модель нужно включить краткое описание каждого актера или прецедента, делается это для того, чтобы между разработчиком и заказчиком системы не оставалось «белых пятен» и расхождений в понимании функциональности системы и ролей взаимодействующих с ней актеров. Для каждого актера описывается роль, которую он

играет в системе, а для каждого прецедента – его назначение и функциональность. Также можно уточнить, каким актером запускается прецедент.

В StarUML добавление описания к элементам модели делается следующим образом. Выделите элемент модели, щелкнув по нему мышкой, и откройте редактор Documentation. Если он не отображается справа на одной из вкладок инспектора модели, то его можно открыть, используя меню View→Documentation. Напротив пункта Documentation должна стоять галочка. Далее необходимо ввести описание элемента в окно документирования, как показано на рис. 3.16.

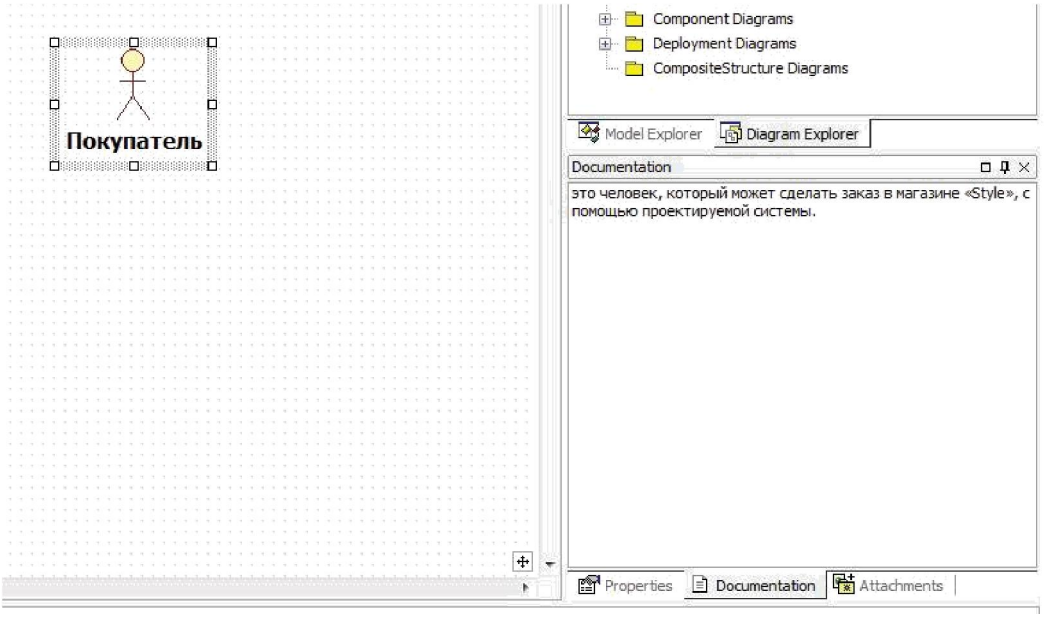


Рисунок 3.16 – Документирование элемента модели

Все элементы модели должны быть задокументированы. Описанный выше способ подходит для любого элемента любой диаграммы.

Для актеров и прецедентов системы заказов магазина «Style» можно использовать следующее описание:

- Покупатель – это человек, который может сделать заказ в магазине «Style», с помощью проектируемой системы.
- Сотрудник – это все сотрудники магазина «Style», которые могут получать информацию о сделанных заказах и изменять статус заказа в системе в зависимости от того шага, на котором находится обработка данного заказа.
- Система «Склад» – это внешняя система, которая получает информацию о сделанных в магазине «Style» заказах для того, чтобы обеспечить учет наличия товаров на складе и снабжение товарами.
- Заказ товаров – этот прецедент запускается покупателем для того, чтобы оформить заказ в магазине «Style». Состоит из просмотра каталога, добавления товаров в корзину, просмотра корзины, изменения содержания корзины и оформления заказа, включая оплату.
- Управление статусом заказа – этот прецедент используется сотрудниками магазина для изменения статуса заказа в процессе его обработки.
- Получение информации о заказе – прецедент используется всеми актерами для просмотра информации о заказе.

Для того чтобы создать еще одну диаграмму (любого типа), например, детализирующую прецедент, необходимо щелкнуть правой кнопкой мыши по папке Use Case Model и в появившемся контекстном меню выбрать Add Diagram, затем из списка диаграмм выбрать требуемую.

Наиболее значимым для данной системы и ее актеров прецедентом является прецедент Заказ товаров. Для него мы построим дополнительную диаграмму прецедентов, поясняющую этот вариант использования, рис. 3.17.

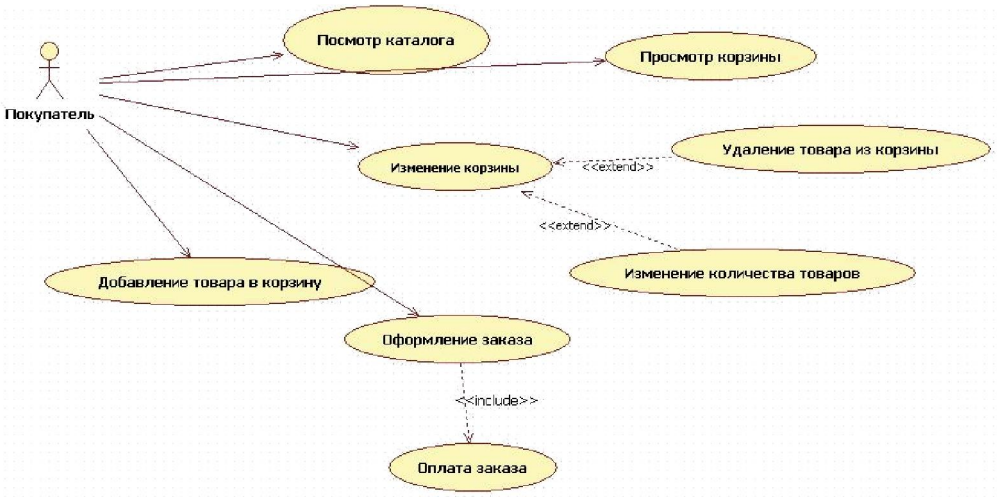


Рисунок 3.17 – Диаграмма вариантов использования, поясняющая основной прецедент Заказ товаров

Одним из требований языка UML является самодостаточность диаграмм для представления информации о моделях проектируемых систем. Однако, как уже отмечалось выше, диаграммы вариантов использования описывают то, что делает система, без уточнения того, как она это делает.

Для реального описания системы потребуются более специфические данные, которые отражены в потоке событий. Потоки событий уточняют или детализируют последовательность действий, совершаемых системой при выполнении ее вариантов использования, а также описывают логику переходов через варианты использования.

Поток событий – это определенная последовательность действий, которая описывает действия актеров и поведение моделируемой системы в форме обычного текста.

Потоки событий являются текстовым описанием пошагового выполнения прецедентов, они понятны не только разработчику, но и стороннему читателю. Их задача – еще больше детализировать описание функциональности системы до того, как разработчики приступят к написанию программного кода, и устранить возможное недопонимание требуемой функциональности, как можно больше сблизить представления разработчика о системе и заказчика.

Потоки событий бывают трех типов: основной, альтернативный и поток ошибок:

- Основной (главный) поток описывает наилучший сценарий либо наиболее используемый путь исполнения прецедента.
- Альтернативный поток специфицирует отклонения от основного потока, которые не рассматриваются как ошибочные.
- Поток ошибок рассматривается как отклонение от альтернативного или основного, которое порождает условия формирования ошибки.

Опишем потоки событий прецедента Заказ товаров.

Основной поток событий

1 Прецедент начинается с выбора покупателем режима показа каталога.
2 Система открывает каталог.
3 Покупатель выбирает режим показа корзины.

А1. Покупатель просматривает каталог и запускает поток «добавление товара в корзину»

4 Система открывает корзину.
5 Покупатель нажимает кнопку «оформить заказ».

A2. Покупатель просматривает корзину и запускает поток «изменение корзины».

A3. Покупатель решает вернуться в каталог.

6 Система переходит к первому шагу оформления заказа: запрашивает у покупателя личные данные и телефон.
7 Покупатель вводит личные данные и телефон.
8 Система переходит ко второму шагу оформления заказа: показывает содержимое заказа и просит подтвердить заказ.
9 Покупатель подтверждает заказ.

A4. Покупатель возвращается в корзину.

10 Система переходит к третьему шагу оформления заказа: запрашивает тип кредитной карты, ее номер, секретный код, имя владельца и дату завершения срока действия.
11 Покупатель вводит тип кредитной карты, ее номер, секретный код, имя владельца и дату завершения срока действия.
12 Система переходит к четвертому шагу оформления заказа: подтверждает оплату.

A5. Счет пользователя не найден.

A6. Недостаточно денег на счете.

E1. Платежная система недоступна.

13 Система присваивает заказу номер и отправляет его вместе подтверждением заказа на электронный адрес покупателя.
14 Вариант использования завершается.

Альтернативные потоки

A1. добавление товара в корзину

1 Покупатель выбирает размер.
2 Покупатель выбирает количество.
3 Покупатель нажимает кнопку Добавить в корзину
4 Система помещает выбранный товар в корзину.

A7. Покупатель не выбрал размер.

A8. Покупатель не выбрал количество.

5 Система выводит сообщение о том, что товар добавлен в корзину.
6 Поток возвращается на второй этап основного потока.

A2. Изменение корзины

1 Покупатель нажимает кнопку «Удалить» напротив одного выбранного товара.

A9. Покупатель изменяет количество позиций одного выбранного товара.

2 Система удаляет товар из корзины.
3 Поток возвращается к этапу 4 основного потока.

A3. Покупатель решает вернуться в каталог

1 Поток возвращается к этапу 2 основного потока.

A4. Покупатель возвращается в корзину

1 Поток возвращается к этапу 4 основного потока.

A5. Счет пользователя не найден

1 Система выводит сообщение о том, что счет пользователя не обнаружен.
2 Поток возвращается к этапу 11 основного потока.

A6. Недостаточно денег на счете

1 Система выводит сообщение о том, что на счете пользователя недостаточно денег для совершения операции.

2 Поток возвращается к этапу 11 основного потока.

A7. Покупатель не выбрал размер

1 Система выводит сообщение о необходимости выбора размера.
2 Поток возвращается на 1 этап потока A1.

A8. Покупатель не выбрал количество

1 Система выводит сообщение о необходимости выбора количества товара.
2 Поток возвращается на 2 этап потока A1.

A9. Покупатель изменяет количество позиций одного выбранного товара

1 Покупатель увеличивает количество товара.
2 Система обновляет корзину.

A10. Недостаточно товара в наличии.

3 Поток возвращается к этапу 4 основного потока.

A10. Недостаточно товара в наличии

1 Система выводит сообщение о том, сколько позиций может заказать покупатель.
2 Поток возвращается к этапу 4 основного потока.

Потоки ошибок

E1. Платежная система недоступна

1 Система выводит сообщение о недоступности платежной системы.
2 Поток возвращается к этапу 11 основного потока.

Чтобы добавить поток событий к модели, нужно выделить прецедент, который он детализирует (в данном примере это прецедент Заказ товаров), в инспекторе модели открыть редактор вложений Attachments, нажать на значок  , расположенный в верхней части редактора вложений, в появившемся окне нажать кнопку  и выбрать соответствующий файл, содержащий описание потока событий. Чтобы удалить вложенный файл, выделите его на редакторе вложений и нажмите кнопку  (рис. 3.18).

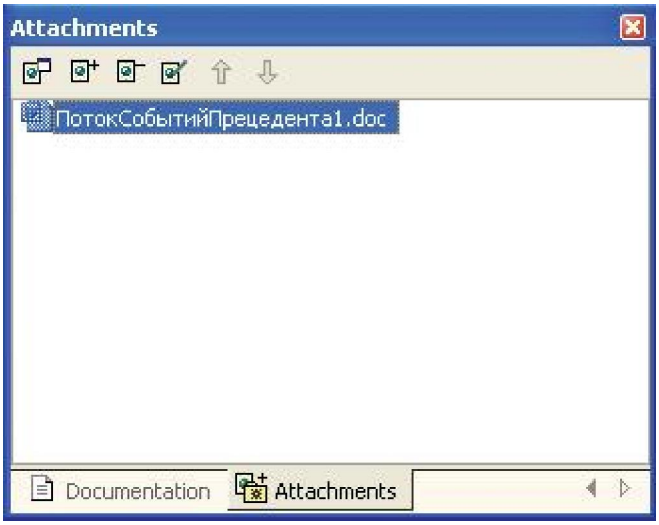


Рисунок 3.18 – Добавление вложений в модель

3.2 Диаграммы деятельности и классов

Диаграммы деятельности обеспечивают еще один способ моделирования потока событий. С помощью текстового описания можно рассказать о потоке, но трудно будет понять логику событий в сложных и запутанных потоках с множеством альтернативных

ветвей. Диаграммы деятельности создаются также на разных этапах жизненного цикла системы для отражения последовательности выполнения операций.

Рассмотрим основные элементы нотации диаграмм деятельностей. На них иллюстрируются деятельности, переходы между ними, элементы выбора и синхронизации. Деятельностью называется исполнение определенного поведения в потоке управления системы. В UML деятельность изображается в виде скругленного прямоугольника с текстовым описанием внутри.

Деятельность обозначает некоторый шаг (этап) процесса. В прецеденте Заказ товаров одним из таких шагов может быть Добавить товар в корзину, рис. 3.19.

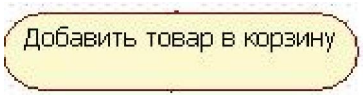


Рисунок 3.19 – Деятельность

Переход показывает, как поток управления переходит от одной деятельности к другой. Обычно переход осуществляется по завершении деятельности (рис. 3.20). В рассматриваемом примере выполняя Заказ товаров покупатель может Открыть корзину и Удалить товар из нее. Это две разные деятельности, переход к удалению товара возможен только после открытия корзины.

Рисунок 3.20 – Переход между деятельностями

Два состояния на диаграмме деятельности – начальное и конечное определяют продолжительность потока. Начальное состояние обязательно должно быть отмечено на диаграмме, оно определяет начало потока. Конечных состояний может быть несколько или не одного. Оно определяет точку завершения потока. Начальное состояние изображается жирной точкой, а конечное – жирной точкой в окружности (рис. 3.21).

Рисунок 3.21 – Обозначения начального и конечного состояний

При моделировании управляющих потоков системы часто бывает необходимо показать места их разделения на основе условного выбора. Выбор на диаграмме показывается ромбом, помещенным на переходе. Ограничительные условия, от которых зависит выбор направления перехода, помещаются обычно над ромбом. В нотации UML условия записываются в квадратных скобках: [условие].

Если все товары, которые хочет заказать покупатель, добавлены в корзину, то покупатель может просмотреть корзину и оформить заказ. Условие перехода от деятельности Добавить товар в корзину к Просмотреть корзину на диаграмме можно показать так, как это изображено на рис. 3.22.

Рисунок 3.22 – Условие перехода между деятельностями

Синхронизация – способ показать, что две или более ветвей потока выполняются параллельно. Деятельности, помещенные между двумя жирными линиями на диаграмме деятельности, исполняются синхронно, одновременно.

После оплаты заказа покупателем система присваивает заказу уникальный номер и отсылает подтверждение заказа на электронную почту покупателя. Эти две деятельности можно выполнить синхронно, рис. 3.23.

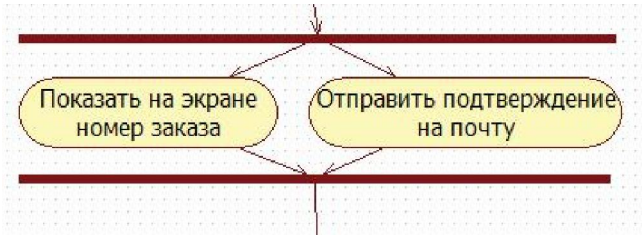


Рисунок 3.23 – Линии синхронизации

Секции делят диаграмму деятельности на несколько участков. Это нужно для того, чтобы показать, кто отвечает за выполнение деятельности и в каком порядке. Если деятельность находится на секции с именем Покупатель, то этот актер и выполняет ее, рис. 3.24.

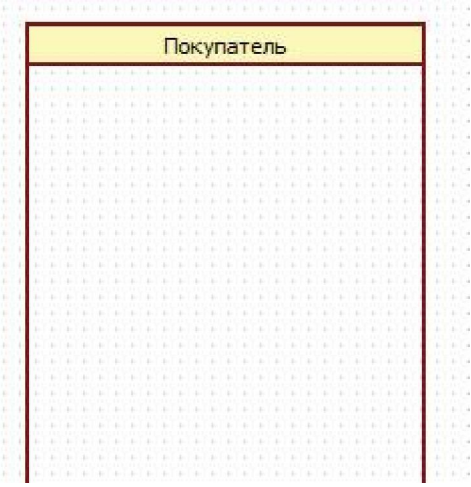


Рисунок 3.24 – Вид секции Покупатель

Теперь рассмотрим процесс создания диаграммы деятельности. Чтобы построить диаграмму деятельности для некоторого прецедента в StarUML, нужно щелкнуть правой кнопкой мыши по этому прецеденту, в выпавшем контекстном меню выбрать пункт Add Diagram, затем в появившемся списке выбрать Activity Diagram.

Поле для создания диаграммы деятельности появится в окне программы, изменится панель инструментов слева, и новая диаграмма отобразится на навигаторе модели.

Построим диаграмму деятельности для дополнительного прецедента Оформить заказ актера Покупатель, рис. 3.25. Оформление заказа включает указание своих личных контактных данных, электронной почты и оплату заказа. Так, оформление начинается из корзины покупателя, когда он выбирает опцию «Оформить заказ».

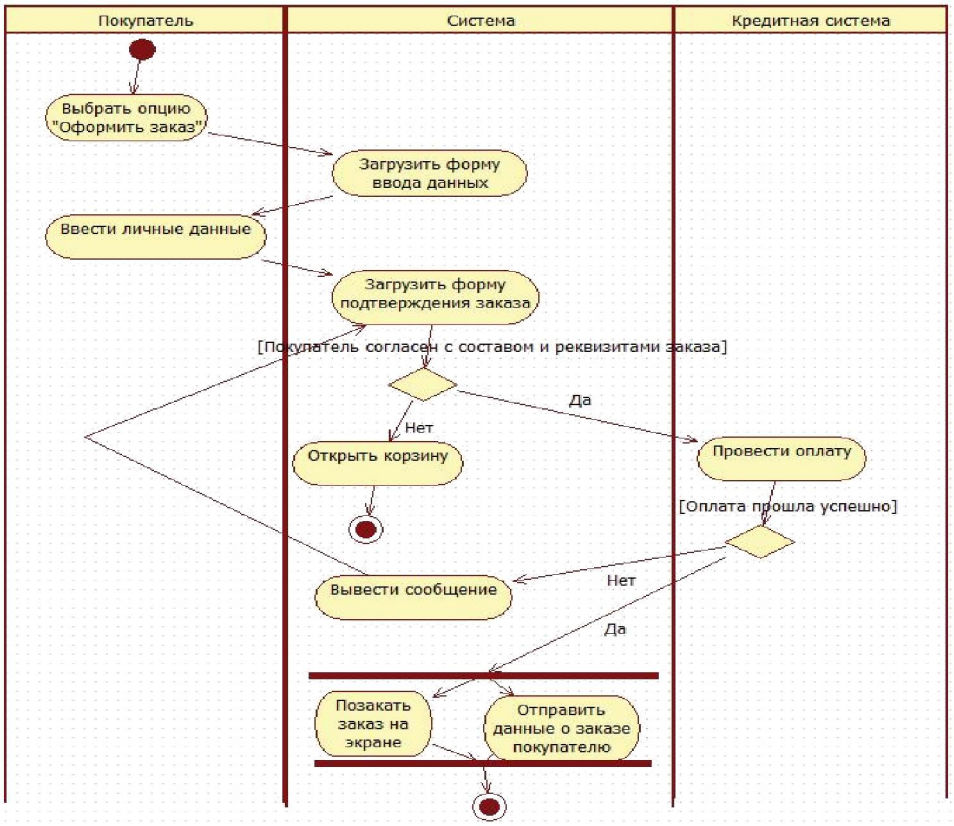


Рисунок 3.25 – Диаграмма деятельности прецедента Оформить заказ

Далее рассмотрим процесс создания диаграммы классов.

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

Дадим некоторые определения и опишем основные элементы нотации диаграмм классов:

Объект – это некоторая сущность реального мира или концептуальная (абстрактная) сущность.

Примерами объектов могут служить дом №4 по улице Садовая, сотрудник фирмы Иван Петров, ваш компьютер. Или нечто абстрактное: химическая формула, торговый заказ номер 456789, банковский счет клиента Петра Иванова. 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.

Покупатель определяется его именем, адресом, телефоном, датой рождения.

Поведение – определяет, как объект реагирует на запросы других объектов и что может делать сам объект. Поведение характеризуется операциями объекта.

Покупатель может добавить товар в корзину, просматривать каталог, удалять товар из корзины.

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

Пример. Объекты Мария Петрова и Анна Седова уникальны, хотя каждый из них является покупателем магазина и имеет одинаковые поведение и состояния.

Как правило, в системе существует множество объектов имеющих одинаковое поведение, принимающих одинаковые состояния. Например, сотрудники фирмы, которых может быть несколько десятков, и данные о которых содержатся в базе данных, имеют одинаковые атрибуты – фамилию, имя, отчество, дату рождения, должность и др. – с разными значениями этих атрибутов, а также могут иметь схожее поведение – подать заявление на отпуск или перевод в другое подразделение. Для группировки объектов используются классы.

Класс – это описание группы объектов с общими свойствами (атрибутами), поведением (операциями), отношениями с другими объектами и семантикой.

Каждый класс является шаблоном для создания объекта. А каждый объект – это экземпляр класса. Важно помнить, что каждый объект может быть экземпляром только одного класса.

Применительно к магазину «Style» необходимо сгруппировать сотрудников магазина, описав общий для них класс Сотрудник. Объект этого класса, например, Иван Петров, может включать в себя следующую информацию: имя, адрес, должность, размер заработной платы, кроме того этот объект может выйти в отпуск.

В нотации UML классы и объекты изображаются в виде прямоугольников, рис. 3.26. Прямоугольник класса всегда делится на три секции (раздела), имя класса помещается в первую секцию, каждое слово в названии класса принято писать с большой буквы. Во второй и третьей секциях могут указываться атрибуты и операции класса соответственно, эти секции могут быть пустыми. Названия классов выбираются в соответствии с понятиями предметной области. Это должно быть существительное или словосочетание в единственном числе, наиболее точно характеризующее предмет. Класс должен описывать только одну сущность.

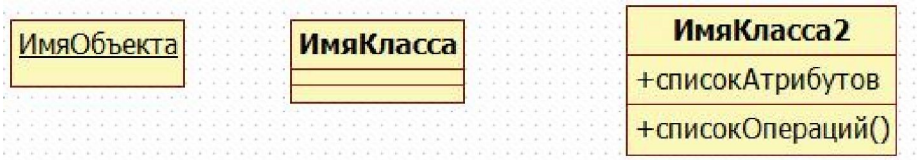


Рисунок 3.26 – Изображение классов и объектов

Имя класса может быть простым (рис. 3.26) или составным. Составное имя класса состоит из самого имени класса и из имени пакета, которому принадлежит класс, разделенных двоеточием, рис. 3.27. Имя класса должно быть уникальным внутри пакета.

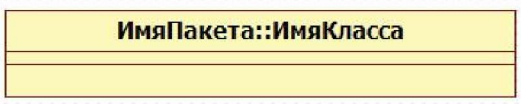


Рисунок 3.27 – Составное имя класса

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. 3.28.

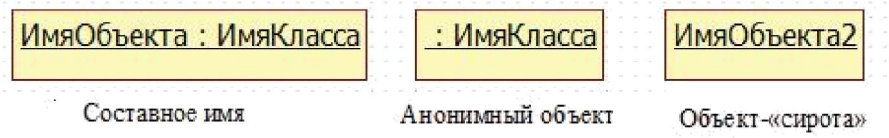


Рисунок 3.28 – Именование объектов

Класс Сотрудник и объект этого класса – некоторого сотрудника можно изобразить так, как показано на рис. 3.29. В данном случае объекту класса Сотрудник присвоено имя, совпадающее с именем класса.

Рисунок 3.29 – Класс и его объект

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 построена диаграмма взаимодействия, перед тем, как приступать к построению диаграмм классов, лучше начать с поиска на этих диаграммах похожих объектов. Пусть диаграмма последовательности, описывающая оформление заказа объектами Ивановым и Петровым. Если посмотреть 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.

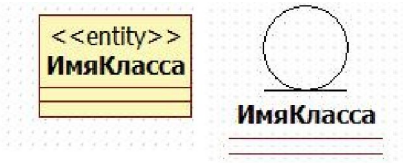
Некоторые возможные классы будут выявлены при рассмотрении трех стереотипов: сущность (entity), граница (boundary) и управление (control). Ранее рассматривались стереотипы отношений, когда говорилось об отношениях на диаграммах прецедентов. Тот же принцип создания нового типа на основе уже существующего применим и для классов.

Стереотип – это механизм, позволяющий категоризировать классы. Он используется для создания нового типа элемента, в данном случае нового типа класса.

Например, есть необходимость 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. 3.30.



Риунок 3.30 – Обозначение классов-сущностей

Граничными классами называются классы, расположенные на границе системы со всем остальным миром, и т.о. 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. 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. 3.31.

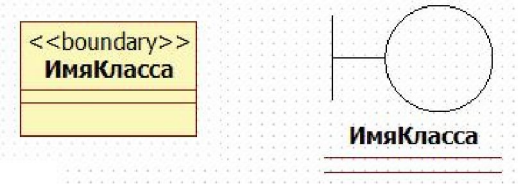


Рисунок 3.31 – Обозначение граничных классов

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. 3.32.

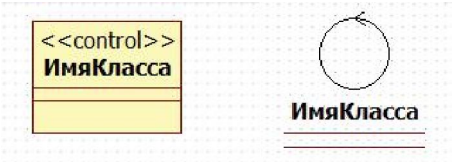


Рисунок 3.32 – Обозначение управляющих классов

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 актер/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. Так, поскольку в рассматриваемой модели присутствует класс Сотрудник, то хорошим описанием для него будет:

Сотрудник – это человек, работающий на фирме. Класс содержит информацию, необходимую для исполнения организацией своих обязанностей по отношению к сотруднику (начисление зарплаты, перевод на другую должность, увольнение и т.п.).

Плохим описанием будет описание структуры класса, которая может быть и так описана с помощью атрибутов. Например, плохое описание класса Сотрудник: Имя, телефон, адрес, должность, зарплата.

В StarUML документирование классов выполняется также как и описанное выше документирование прецедентов. Нужно выделить класс, который необходимо описать, открыть окно документирования Documentation на инспекторе модели и ввести описание класса.

Диаграммы классов относятся к логическому представлению системы Logical View. На диаграмме Main представления Logical View обычно размещают главную диаграмму пакетов, а диаграммы классов помещают на другие листы этого представления. 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, рис. 3.33.

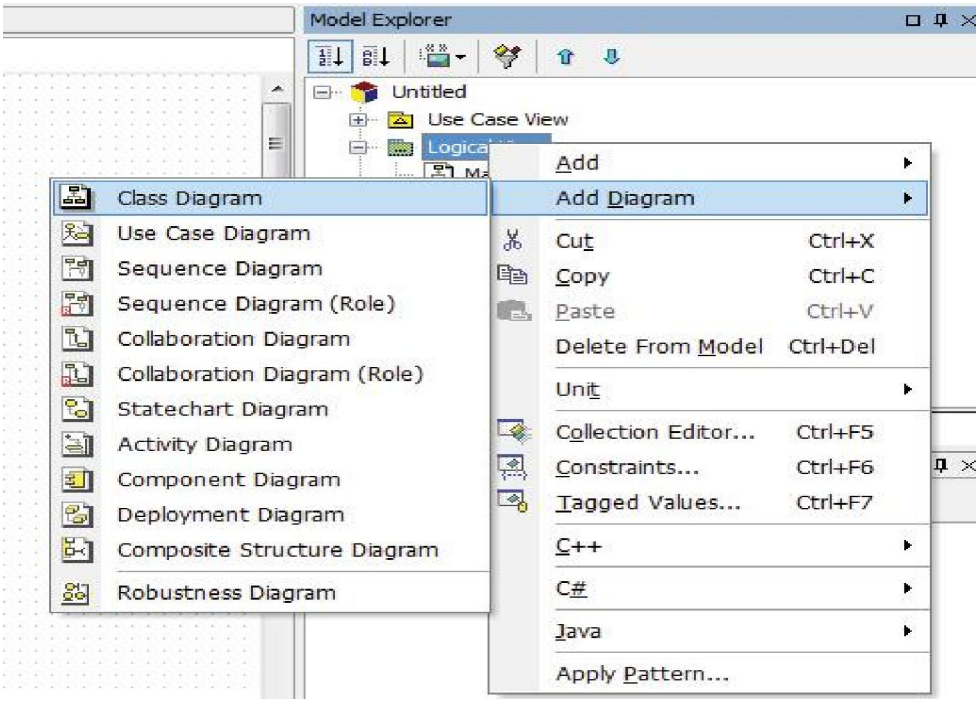


Рисунок 3.33 – Добавление диаграммы классов

Будет создана новая диаграмма классов со стандартным именем ClassDiagram1, которое можно изменить в редакторе свойств диаграммы, рис. 3.34.

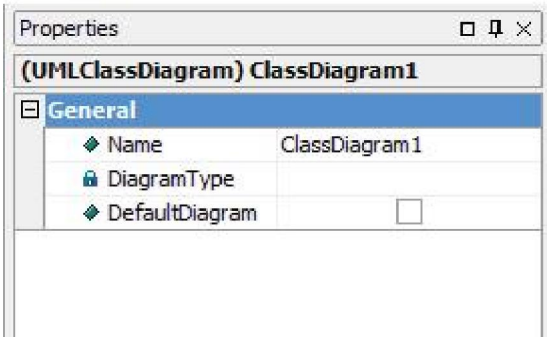


Рисунок 3.34 – Редактор свойств диаграммы классов: изменение имени диаграммы

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.

Представим, 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 ОформлениеЗаказа (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 введем еще один класс ВводЛичныхДанных (EnterPersonalInformation). 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 ПроверкаДеталейЗаказа (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 ПодтверждениеЗаказа (OrderConfirmation)

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 МенеджерОформленияЗаказа (PlaceOrderManager).

Выбор классов-сущностей. В сценарии Оформление заказа речь идет о покупателе, заказе и товарах. Создадим классы-сущности Покупатель (Customer), Заказ (Order), Товар (Item).

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

Построим диаграмму классов сценария Оформление заказа. Для этого необходимо создать в пакете Logical View диаграмму классов описанным выше способом. Далее переименовать ClassDiagram1 в Place Order (Оформление Заказа).

Создадим новый класс Покупатель (Customer). Щелкнув правой кнопкой мыши по Logical View в навигаторе модели, в контекстном меню выбрать пункт Add (Добавить), затем выбрать пункт Class (Класс). Новый класс будет создан и отобразится в навигаторе модели. На вкладке Properties (Свойства) необходимо изменить его имя на Покупатель (Customer). При таком способе создания класса, StarUML создает новое пространство имен. Если необходимо создать класс, который носит такое же имя, как, например, действующее лицо на диаграмме прецедентов, то для того, чтобы StarUML «разрешил»

использовать это имя еще раз, нужно создать класс способом, описанным выше. Далее требуется перетащить этот класс из навигатора модели на лист диаграммы Place Order, рис. 3.35.

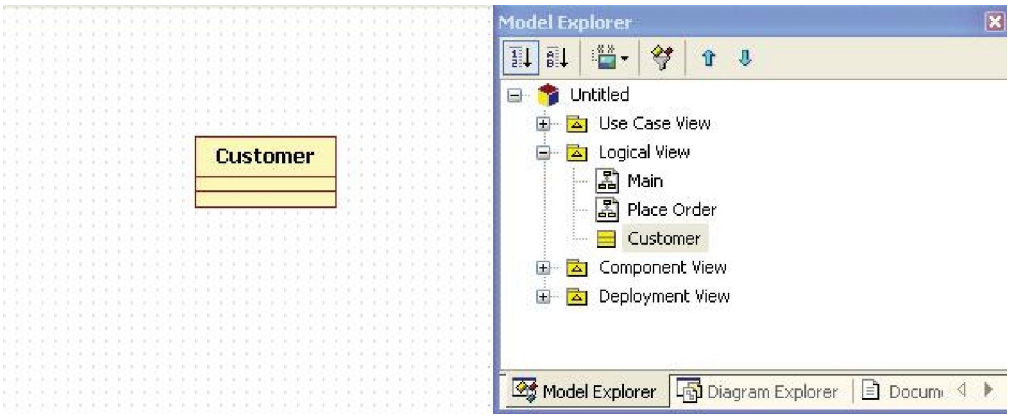


Рисунок 3.35 – Создание класса Покупатель

Для того чтобы создать класс ОформлениеЗаказа (PlaceOrder), воспользуемся другим методом. Для этого слева на панели инструментов (Toolbox) необходимо щелкнуть изображение класса (Class), затем щелкнуть на листе диаграммы Place Order в том месте, куда необходимо поместить класс, при этом стандартное имя класса станет активным, давая возможность его изменить. Имя класса необходимо сменить на ОформлениеЗаказа (PlaceOrder), рис. 3.36.

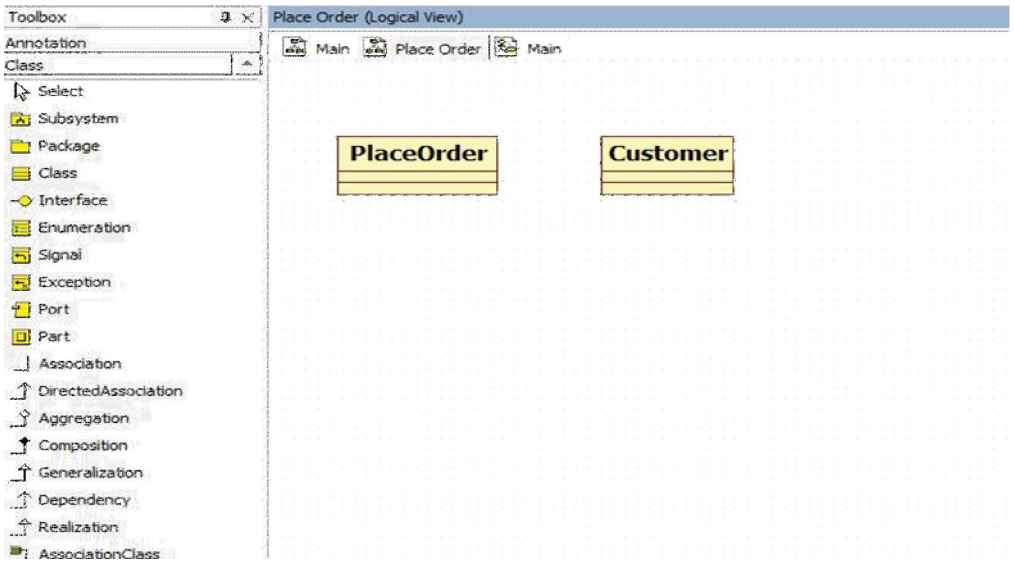


Рисунок 3.36 – Создание класса ОформлениеЗаказа

Далее необходимо создать остальные классы. В итоге диаграмма примет вид как показано на рис. 3.37.

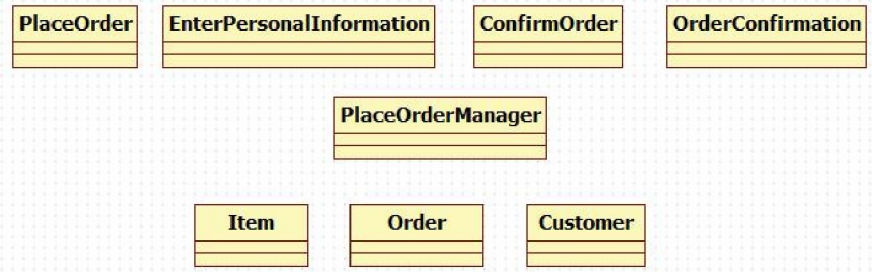


Рисунок 3.37 – Диаграмма классов сценария Оформление заказа

Чтобы присвоить классу один из стереотипов UML, нужно выделить класс щелчком правой кнопки мыши, открыть редактор свойств Properties на инспекторе модели и выбрать раздел стереотип Stereotype, рис. 3.38.

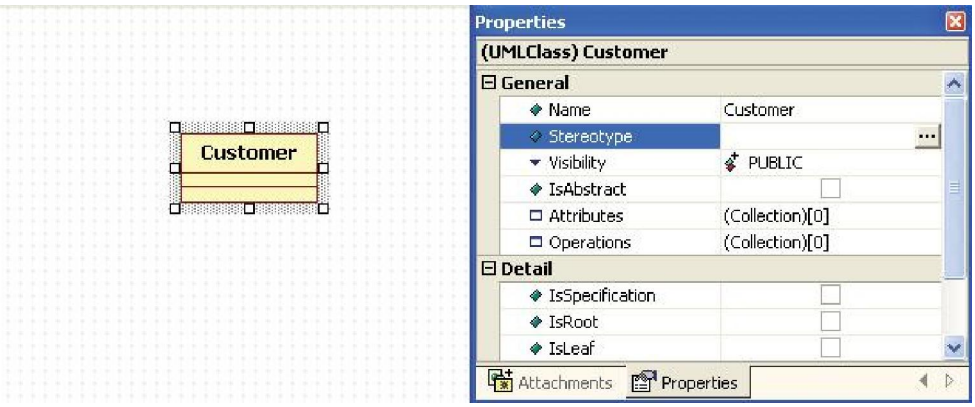


Рисунок 3.38 – Диаграммы взаимодействия и кооперации

Если нажать на значок  , то в появившемся диалоге будет представлен список доступных стереотипов с кратким описанием и графическим обозначением. Для того чтобы присвоить классу стереотип, нужно выбрать его в списке, выделить и нажать кнопку OK, рис. 3.39.

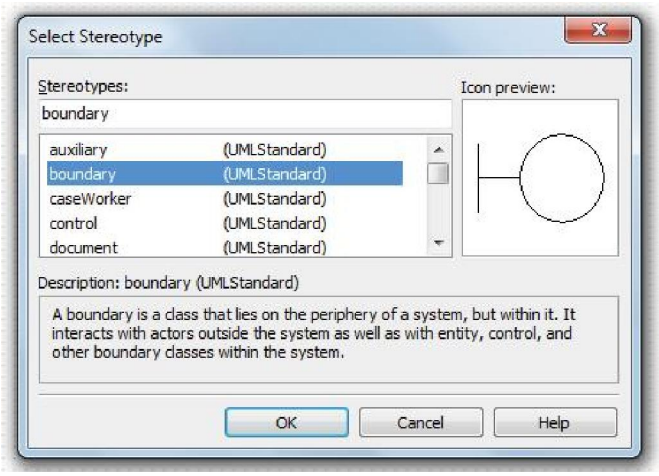


Рисунок 3.39 – Выбор стереотипа

После присвоения классу стереотипа его внешний вид изменится. Рядом с именем класса появится имя стереотипа, заключенное в угловые скобки, рис. 3.40.

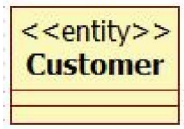


Рисунок 3.40 – Отображение стереотипа класса

Далее необходимо присвоить классам сценария Оформление заказа соответствующие стереотипы. В итоге диаграмма классов изменится, рис. 3.41.

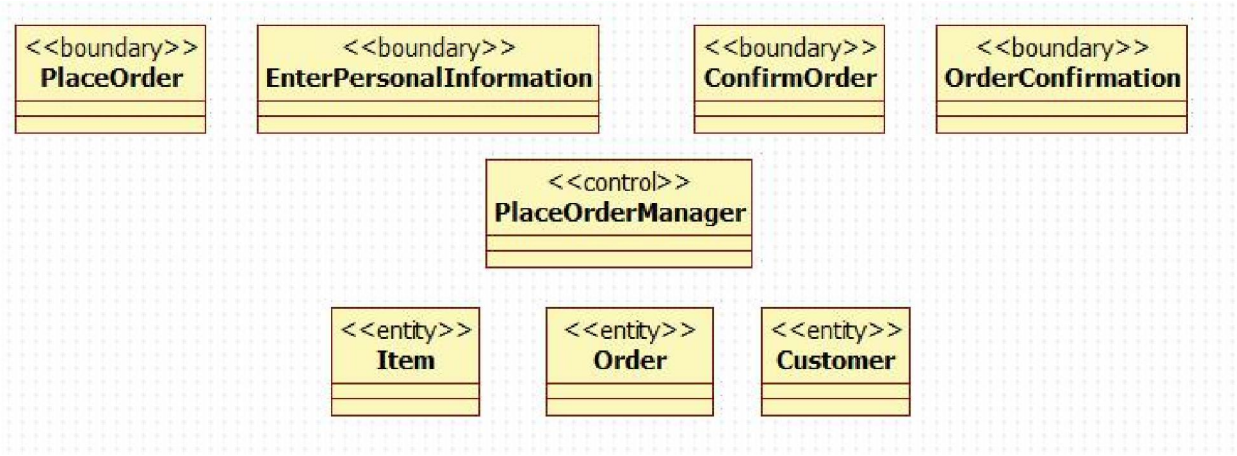


Рисунок 3.41 – Диаграмма классов сценария Оформление заказа со стереотипами

Отобразить стереотипы классов можно с помощью пиктограмм. Для этого нужно выделить класс, щелкнуть по выделенной области правой кнопкой мыши, в контекстном меню выбрать пункт Format, затем пункт Stereotype Display и далее в списке выбрать Iconic, как показано на рис. 3.42. После чего классы будут отображаться как пиктограммы.

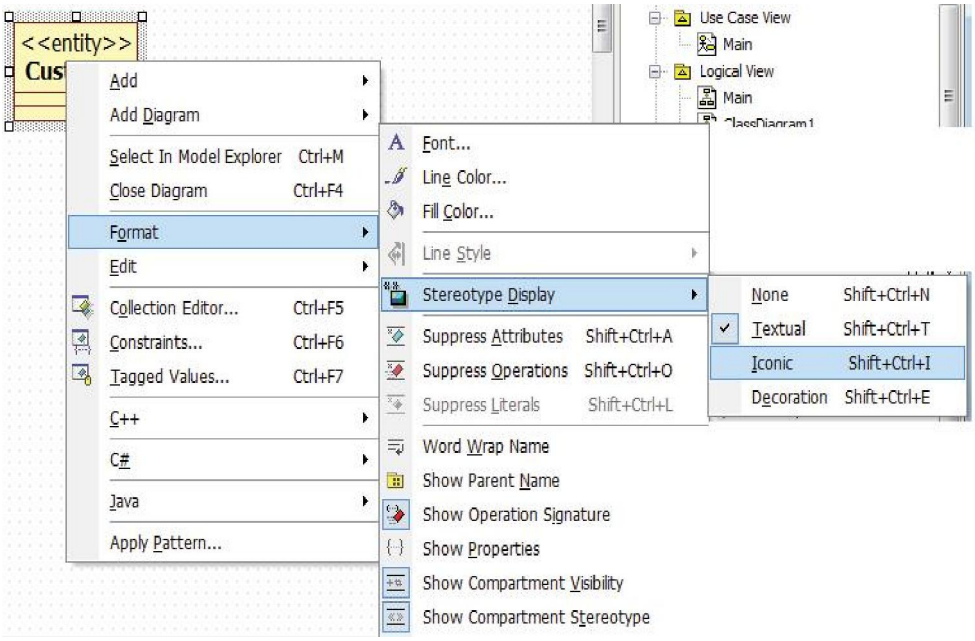


Рисунок 3.42 – Задание отображения стереотипов классов в виде пиктограмм

На рис. 3.43 приведена диаграмма классов сценария Оформление заказа с помощью пиктограмм.

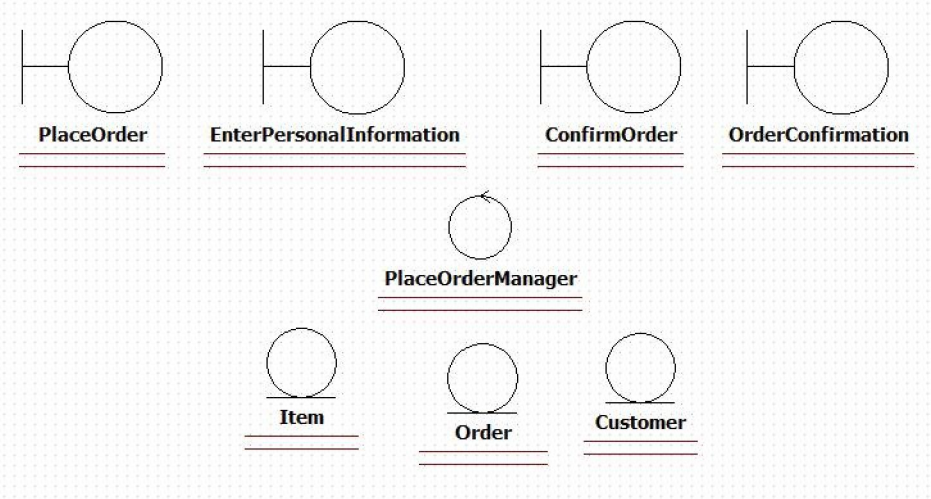


Рисунок 3.43 – Диаграмма классов сценария Оформление заказа в пиктограммах

Если в рассматриваемой 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 являются 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 можно получить представление о системе на более высоком уровне. Напротив, рассматривая пакет, можно более детально представить модель.

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

1 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 комбинация предыдущих двух подходов.

В дальнейшем можно вкладывать пакеты друг в друга. Чаще всего пакет на диаграмме изображается в виде папки с закладкой с именем пакета (рис. 3.44).

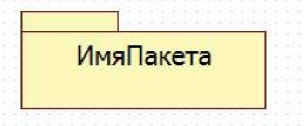


Рисунок 3.44 – Пакет

Главную диаграмму пакетов системы обычно помещают диаграмме Main представления Logical View. 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 куда необходимо 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.

Созданные ранее классы сценария Оформление заказа сгруппируем по пакетам, диаграмму пакетов поместим на листе Main представления Logical View. Создадим пакеты Граничные классы, Классы-сущности, Управляющие классы (рис. 3.45).

Рисунок 3.45 – Диаграмма пакетов

Для включения класса в пакет его необходимо просто переместить в навигаторе модели. Так, на рис. 3.46 показан пример включения класса Customer (Покупатель) в пакет Классы-сущности.

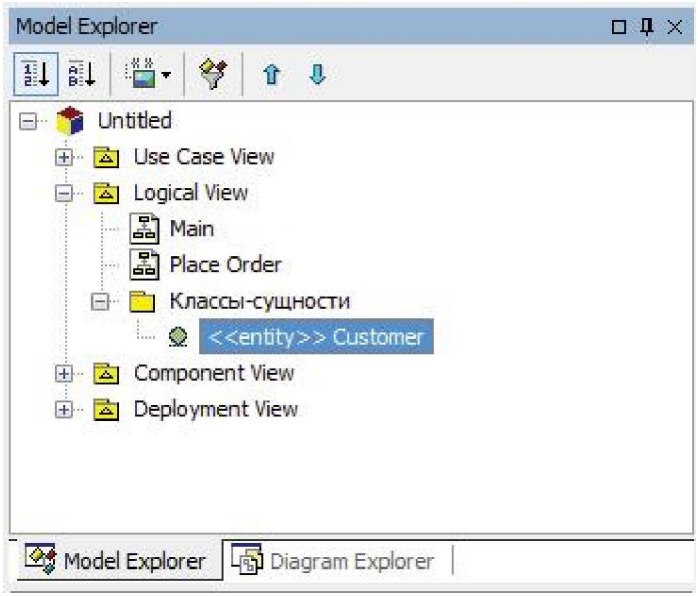


Рисунок 3.46 - Перемещение классов в пакеты

Аналогично классы Item (Товар) и Order (Заказ) необходимо включить в пакет Классы-сущности, классы ОформлениеЗаказа (PlaceOrder), ВводЛичныхДанных (EnterPersonalInformation), ПроверкаДеталейЗаказа (ConfirmOrder) и ПодтверждениеЗаказа (OrderConfirmation) - в пакет Граничные классы, а МенеджерОформленияЗаказа (PlaceOrderManager) – в пакете Управляющие классы.

Стоит отметить, что внешний вид диаграмм классов Place Order и диаграммы пакетов Main при этом не изменится.

3.3 Диаграммы взаимодействия и кооперации

Диаграммы взаимодействия отображают один из процессов обработки информации в варианте использования: какие объекты нужны потоку, какими сообщениями обмениваются объекты, какие действующие лица инициируют поток и в какой

последовательности отправляются сообщения. Для одного потока событий может быть построено несколько диаграмм взаимодействия.

Основной элемент диаграмм взаимодействия – это объект.

Объектом описывают нечто содержащее в себе данные и поведение. Это термин, описывающий реальные, конкретные предметы или абстрактные сущности. Они изображаются в виде прямоугольников. Внутри прямоугольника записывается с большой буквы имя объекта, которое подчеркивается. Если оно содержит несколько слов, то все они должны начинаться с большой буквы.

Поскольку рассматриваемая система предназначена для заказа товаров, то необходимо построить диаграмму взаимодействия с участием такого объекта, как Товар (например, диаграмму, описывающую добавление товара в корзину). Можно использовать на диаграмме взаимодействия какой-то конкретный объект-товар, например, Галстук-Бабочка, или можно назвать этот объект безлично – Товар, как показано на рис. 3.47.

Рисунок 3.47 - Примеры именования объектов

Объекты, помещаемые на диаграммы взаимодействия, скорее всего, будут объектами системы и будут относиться к программному обеспечению. При проектировании этих диаграмм можно представлять себе объекты, как экранные формы или части приложений, отвечающие за выполнение определенных действий, или объектом может быть запись в таблице базы данных.

Если перед тем, как строить диаграммы взаимодействия построить диаграммы классов, то тогда поиск объектов упрощается. Объекты соответствуют своим классам или их операциям, и тогда можно создавать и располагать их на диаграмме последовательности действий или кооперативной диаграмме. Если необходимо изучить взаимодействие объектов до того, как переходить к поиску классов, то поиск объектов можно начать с изучения имен существительных в потоке событий. Многие из них станут хорошими кандидатами в объекты. Также можно выделить объекты-сущности, граничные объекты и управляющие объекты на основе выбранных классов.

Существует два типа диаграмм взаимодействия – диаграммы последовательности (или последовательности действий) и диаграммы кооперации. Первые отображают обмен сообщениями между объектами во времени, а вторые отображают структуру взаимодействия. На обеих диаграммах отображается одна и та же информация, но разными способами: диаграмма последовательностей изображает поток управления, а кооперативная диаграмма - поток данных.

Как правило, поток событий описывает не одну последовательность действий, а несколько возможных, это отражается наличием главного потока событий и альтернативных потоков. Чаще всего невозможно описать прецедент с помощью только одной последовательности действий. Например, для прецедента Заказ товаров возможно оформление заказа без изменения корзины, с изменением состава корзины, или покупатель, просмотрев корзину, захочет вернуться в каталог и что-то в нее добавить, возможно, вернувшись в каталог, покупатель не станет ничего больше добавлять, а снова

вернется в корзину и оформит заказ. Каждый такой вариант можно описать своей последовательностью действий, своим сценарием. И, таким образом, один прецедент описывает несколько последовательностей – сценариев, каждый из которых описывает один из вариантов возможного потока событий.

Сценарий – это экземпляр потока событий. Он представляет собой одиночный проход по потоку событий для прецедента. Для графического отображения сценария используются диаграммы последовательностей.

Диаграмма последовательности действий отображает взаимодействие объектов, упорядоченное по времени.На диаграммах последовательности изображаются объекты, классы и последовательность сообщений, которыми обмениваются объекты в ходе выполнения сценария, рис. 3.48.

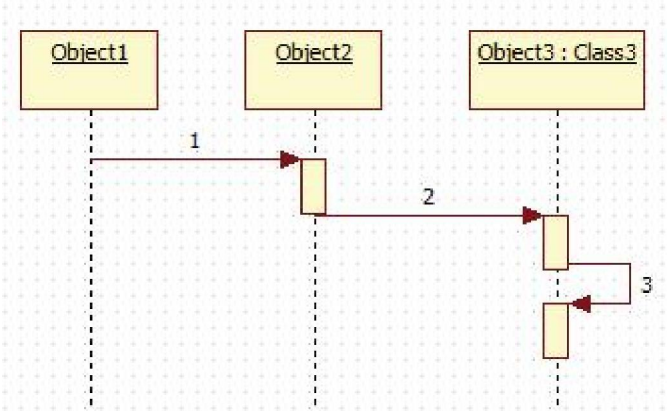


Рисунок 3.48 - Общий вид диаграммы последовательности

На диаграмме последовательностей могут также изображаться экземпляры действующих лиц. Для того чтобы поместить действующее лицо на диаграмму, нужно найти его в навигаторе модели справа и перетащить на поле диаграммы последовательностей. Действующие лица, присутствующие на диаграммах взаимодействия, выделяются из потока событий как сущности, запускающие процессы. На одной диаграмме их может быть несколько. Каждый объект или действующее лицо на диаграмме последовательностей имеет свою линию жизни, которая обозначается пунктиром.

Линия жизни объекта (object lifeline) – вертикальная пунктирная линия на диаграмме последовательности, которая представляет существование объекта в течение определенного периода времени.

Фокус управления (активность, focus of control) - специальный символ на диаграмме последовательности, указывающий период времени, в течение которого объект выполняет некоторое действие, находясь в активном состоянии. Фокус управления изображается тонким прямоугольником, расположенным на линии жизни (рис. 3.48).

Иногда отображение фокуса активности и нумерации сообщений на диаграмме могут сделать ее трудной для чтения. Чтобы фокус управления и нумерация сообщений не отображались на диаграмме последовательности в StarUML нужно открыть редактор свойств этой диаграммы в инспекторе модели и в разделах ShowSequenceNumber и ShowActivation убрать «галочки», рис. 3.49.

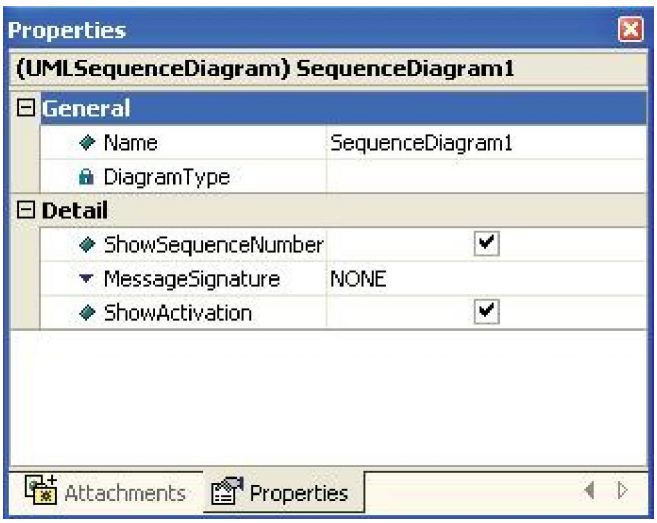


Рисунок 3.49 - Управление отображением фокуса управления и нумерации сообщений

Объекты и действующие лица на диаграммах последовательности обмениваются сообщениями. Сообщения обозначаются стрелками, идущими от отправителя к получателю.

Сообщение (message) - спецификация передачи информации от одного элемента модели к другому с ожиданием выполнения определенных действий со стороны принимающего элемента, рис. 3.50.

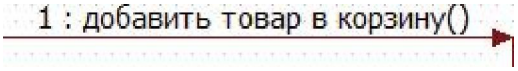


Рисунок 3.50 – Сообщение

Для сообщений на диаграммах последовательностей, как и для других элементов модели, доступен ряд спецификаций. Во-первых, у каждого сообщения должно быть имя, соответствующее его цели. Во-вторых, сообщения на диаграммах последовательностей можно соотнести с операциями, определенными для классов. Если от одного объекта к другому направлено сообщение, то это означает, что объект-источник вызывает операцию объекта-приемника. Объект не может вызвать произвольную операцию: она должна быть доступна этому объекту. В особых случаях сообщение не становится операцией: например, ввод логина и пароля подразумевает их печать в соответствующих полях, и сообщение будет реализовано в виде поля ввода в окне программы. Процедура создания операций из сообщений будет описана ниже. В-третьих, для каждого сообщения можно установить тип синхронизации. Каждому типу соответствует его обозначение.

Вызов операции (процедуры) (call) вызывает операцию того объекта, к которому направлено. Объект может вызвать свою операцию. Тогда стрелка начинается и заканчивается на линии жизни одного и того же объекта, такое сообщение называется рефлексивным (рис. 3.48).

Синхронное сообщение обозначается стрелкой с закрашенным наконечником, рис. 3.51.

Рисунок 3.51- Синхронное сообщение

Асинхронное сообщение (send) посылает объекту сигнал. При этом источник не ждет отклика приемника или подтверждения получения, а продолжает свою работу. Обозначается нежирной стрелкой, рис. 3.52.

Рисунок 3.52 - Асинхронное сообщение

Ответное сообщение (return) возвращает значение из процедуры тому объекту, к которому направлено. Обозначается пунктирной стрелкой, рис. 3.53.

Рисунок 3.53 - Ответное сообщение

Создать объект (create) – создает новый объект. Обозначается стрелкой со стереотипом <<create>>, рис. 3.54.

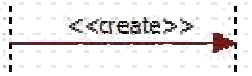


Рисунок 3.54 - Создание объекта

Уничтожить объект (destroy) - удаляет объект. Объект может уничтожить сам себя. Обозначается стрелкой со стереотипом <<destroy>>. При уничтожении объекта на его линии жизни появляется символ разрушения, который обозначается крестом (рис. 63).

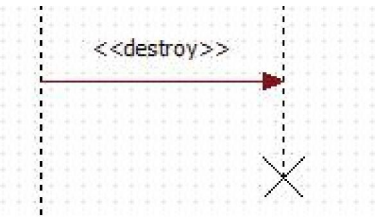


Рисунок 3.55 - Уничтожение объекта

Для определения типа сообщения в StarUML нужно выполнить следующие действия: выделить сообщение, щелкнув по соответствующей стрелке один раз левой кнопкой мыши, после чего откроется редактор свойств, в котором необходимо выбрать раздел ActionKind и в выпадающем списке выбрать необходимый тип синхронизации, рис. 3.56.

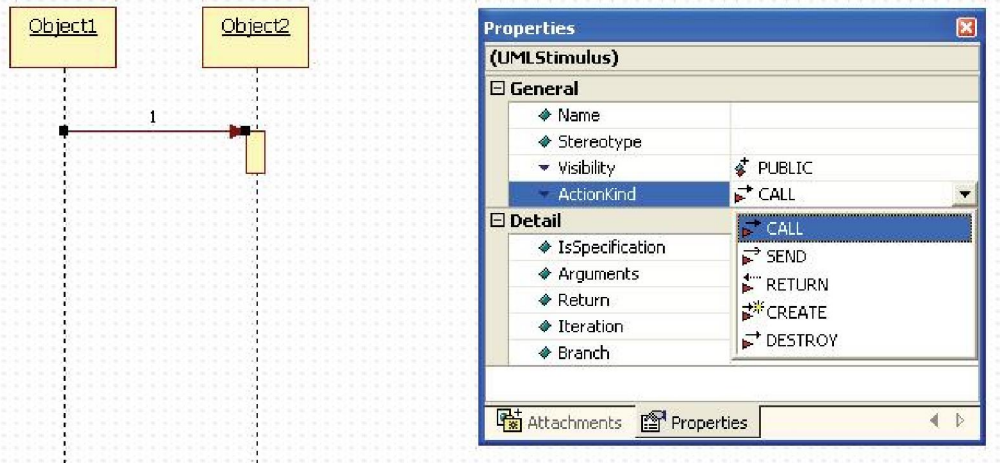


Рисунок 3.56 - Выбор типа сообщения

Для создания новой диаграммы последовательности нужно выполнить 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 последовательности Sequence Diagram, рис. 3.57.

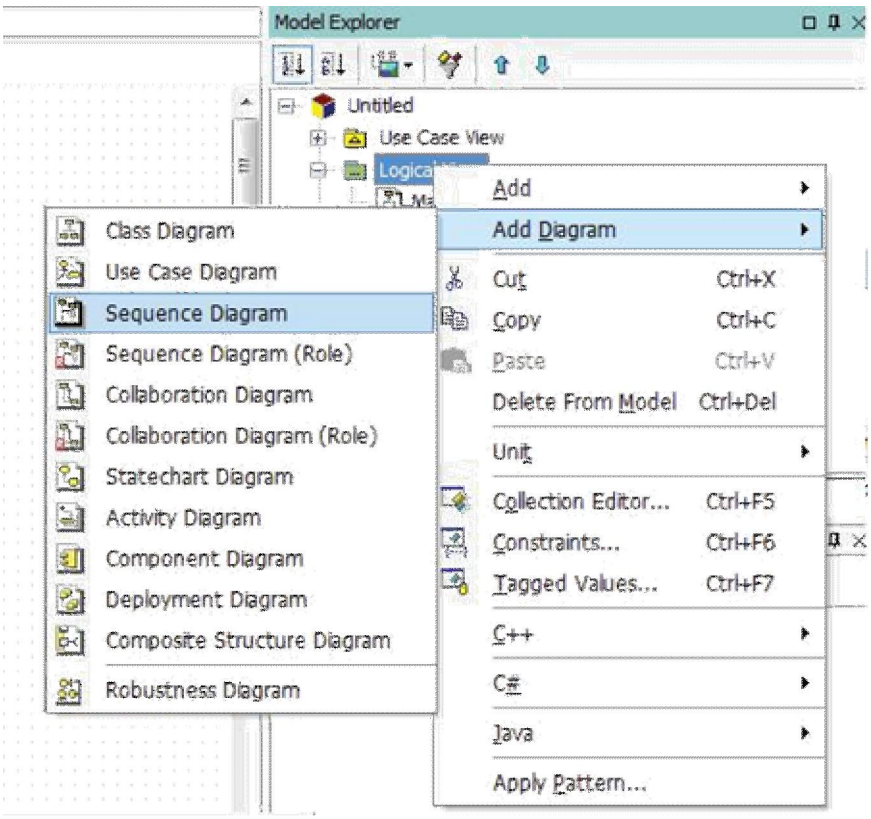


Рисунок 3.57 - Операции над классами и диаграммы состояний

Также можно использовать диаграмму последовательности для детализации прецедента. Для этого нужно связать диаграмму с прецедентом: для создания диаграммы щелкните правой кнопкой мыши по прецеденту, а не по папке Logical View. Однако если строится диаграмма последовательности для анализа системы, то лучше все-таки помещать ее в Logical View.

Поскольку уже определены классы сценария Оформление заказа, теперь с помощью диаграммы последовательности покажем, как взаимодействуют объекты этих классов во времени. Составим диаграмму последовательности для случая, когда покупатель успешно оформляет заказ, рис. 3.58.

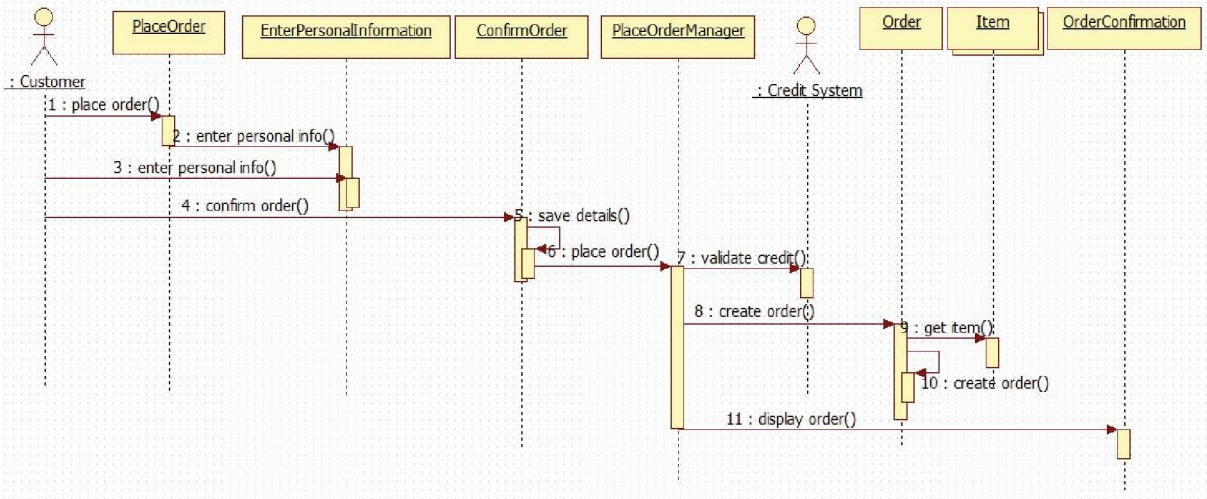


Рисунок 3.58 - Диаграмма последовательности сценария Оформление заказа

Покупатель выбирает опцию «Оформить заказ» (place order), при этом вызывается некоторый объект PlaceOrder (забегая вперед скажем, что это будет граничный объект, принадлежащий соответствующему граничному классу). Далее открывается форма ввода

личных данных покупателя и его кредитной карты (EnterPersonalInformation), на ней покупатель вводит свое имя, адрес, телефон, адрес электронной почты (enter personal information) и кредитные данные. Информация принимается и открывается форма подтверждения заказа (ConfirmOrder), покупатель подтверждает, что согласен с реквизитами заказа (confirm order), детали заказа сохраняются для дальнейшего использования (save the details). Фокус управления передается некоторому управляющему объекту (PlaceOrderManager), который обращается к внешней кредитной системе (Credit System) для проведения платежа. Если платеж прошел успешно (а именно такой сценарий мы сейчас и рассматриваем), то PlaceOrderManager посылает сообщение (create order) создать объект Заказ (Order), затем вызывает форму подтверждения заказа (OrderConfirmation). Объект Заказ (Order) обращается к объектам Товар (Item) для того, чтобы получить информацию о товарах и создает заказ. Процесс завершается.

Стоит отметить, что символ объекта Товар (Item) на диаграмме последовательности отличается от символов других объектов. Дело в том, что задан множественный экземпляр класса. Действительно, заказ может состоять из нескольких товаров, значит объекту Заказ (Order) требуется получить информацию о нескольких объектах Товар (Item). Вместо того чтобы представлять каждый товар отдельно, поэтому использовано представление одним значком несколько объектов.

Чтобы сделать объект множественным в StarUML его необходимо выделить, щелкнув по нему мышью один раз, м в открывшемся редакторе свойств поставьте флажок в разделе IsMultiInstance, рис. 3.59.

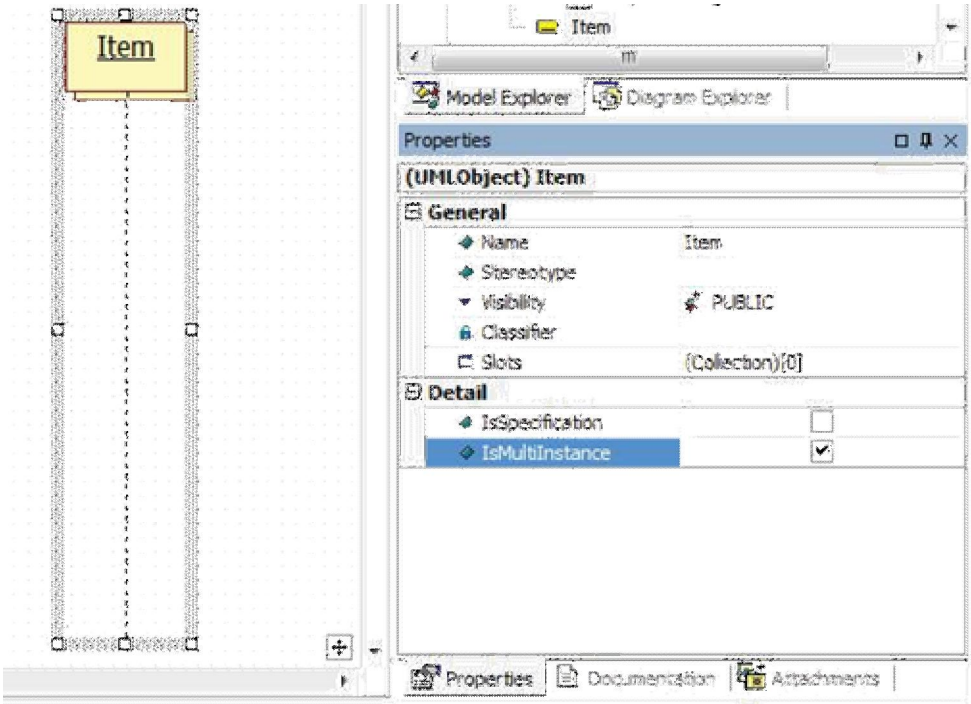


Рисунок 3.59 - Создание множественного объекта

Обычно для основного потока событий большинства прецедентов строится одна диаграмма последовательности, для альтернативных потоков - дополнительные диаграммы, описывающие все остальные сценарии. Так поступают потому, что на диаграмме последовательности действий сложно отобразить логику ЕСЛИ-ТО-ИНАЧЕ. Однако если это необходимо и не загромождает диаграмму, то это можно сделать с помощью условий. Приведем пример.

В процессе оформления покупателем заказа возможны несколько альтернатив. Например, на втором шаге оформления заказа покупатель может подтвердить свой заказ, а может и не согласиться с его реквизитами. На диаграмме последовательности это можно изобразить так, как это показано на рис. 3.60.

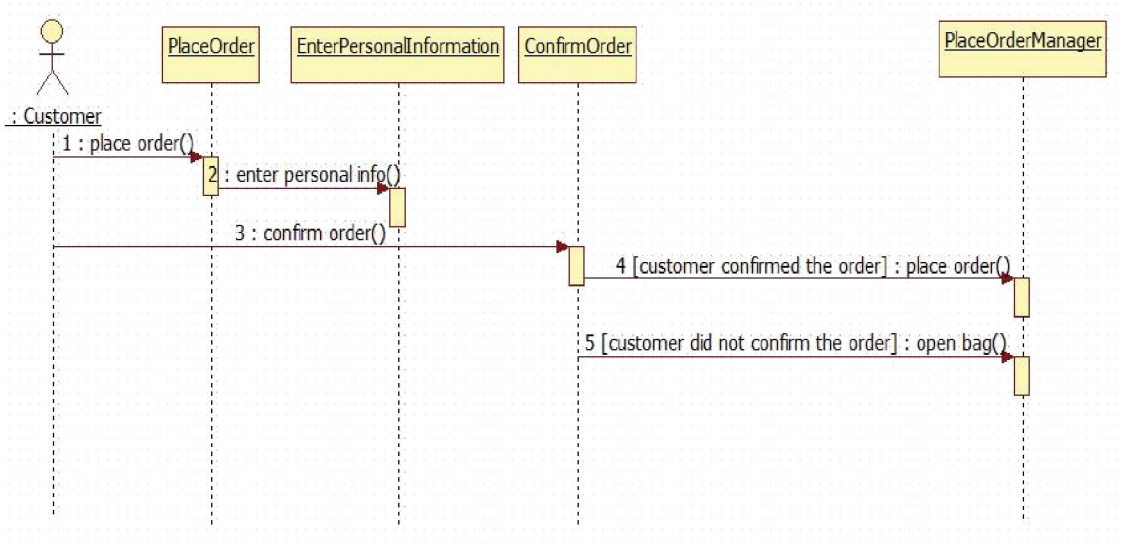


Рисунок 3.60 - Ветвление потока управления

Если покупатель подтверждает свой заказ на втором шаге (customer confirmed the order), то процесс переходит оплате заказа. Если покупатель не подтверждает заказ (customer did not confirm the order), то открывается корзина покупателя. Условие, как это принято в нотации UML, записывается в квадратных скобках [ ]. В данном случае предыдущая диаграмма описания оформления заказа была упрощена, поскольку иначе добавление ветвей процесса сделало бы ее громоздкой и трудно понимаемой. На практике лучше изображать диаграмму последовательности отдельно для каждого сценария потока событий.

Процесс построения модели системы является итерационным. Особенно хорошо это можно видеть при создании диаграмм классов и последовательности. Какую диаграмму создавать первой: классов или последовательности? Одни разработчики начинают с диаграмм классов, другие – наоборот, с последовательности. И в том и в другом случае, скорее всего, обе эти диаграммы, построенные для одного сценария, будут в дальнейшем подвергаться изменению. После построения диаграмм последовательности на диаграммах классов могут появиться новые классы, а на диаграммах последовательности – новые объекты, которых раньше там не было, но они придут туда из диаграмм классов. Возможно, что некоторые объекты и классы будут, напротив, удалены.

В соответствии с построенной диаграммой последовательности для рассматриваемого случая на диаграмме классов сценария Оформить заказ произойдут некоторые изменения. Так, не сложно видеть на диаграмме последовательностей, что покупатель участвует в данном сценарии как действующее лицо-инициатор, запускающий выполнение сценария, но не как внутренний объект системы. Поэтому класс Customer (Покупатель) с данной диаграммы классов можно удалить: скорее всего, такой класс в данной модели все же будет (поэтому его следует удалить только с диаграммы), но классом сценария Оформление заказа он не является. Диаграмма классов прецедента Оформление заказа изменится, и будет выглядеть так, как показано на рис. 3.61.

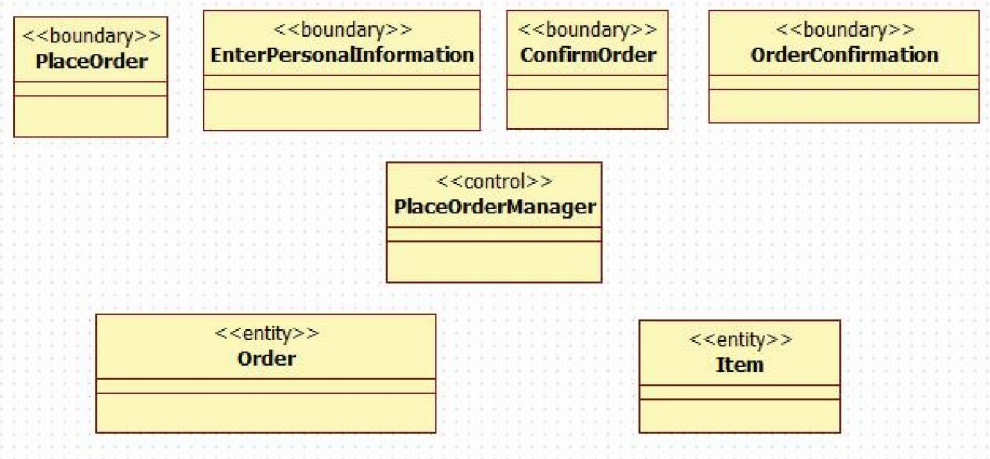


Рисунок 3.61 – Измененная диаграмма классов сценария Оформление заказа

Для создания диаграммы последовательностей можно было не создавать заново каждый объект этой диаграммы, а воспользоваться методом перетаскивания. Если перетащить класс с навигатора модели на диаграмму последовательности, то будет создан анонимный объект этого класса, рис. 3.62. После чего можно изменить имя объекта, рис. 3.63.

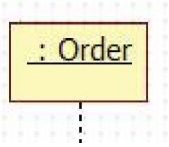


Рисунок 3.62 – Анонимный объект класса Order

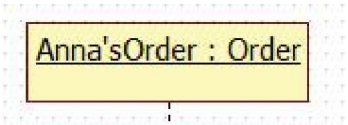


Рисунок 3.63 – Именованный объект класса Order

Далее рассмотрим процесс создания кооперативных диаграмм.

Диаграмма кооперации – это альтернативный способ изображения сценария варианта использования. Этот тип диаграмм заостряет внимание на связях между объектами, отображая обмен данными в системе. В тоже время 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.

Для того чтобы добавить диаграмму кооперации в представление Logical View, необходимо щелкнуть правой кнопкой мыши по папке содержащей диаграмму последовательности (если имя не изменялось, то она называется CollaborationInstanceSet1). Далее в контекстном меню надо выбрать пункт Add Diagram, и в списке выбрать диаграмму кооперации Collaboration diagram, рис. 3.64.

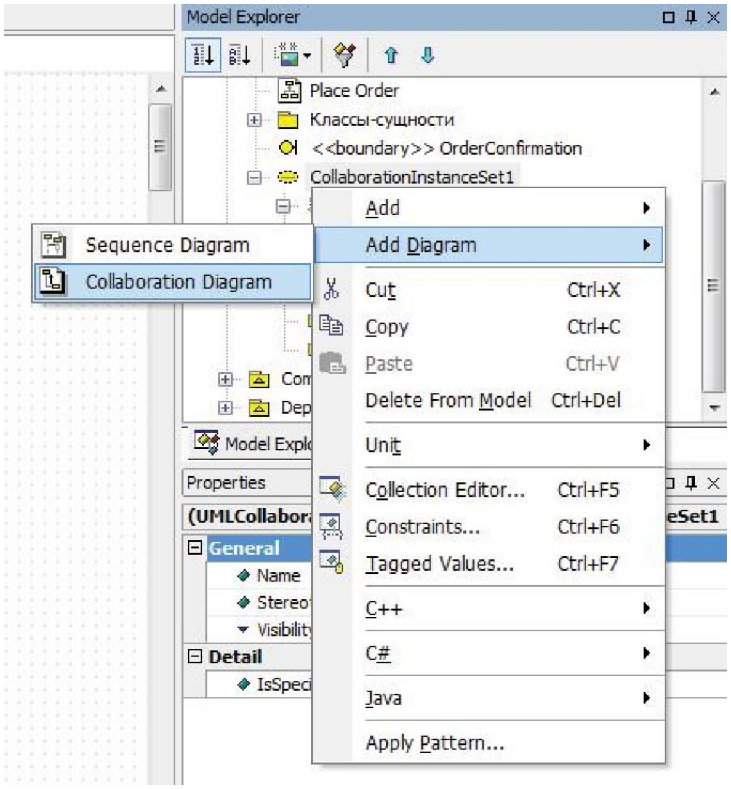


Рисунок 3.64 – Добавление кооперативной диаграммы

Для сценария Оформление заказа, для которого уже составлена диаграмма последовательности, на диаграмму кооперации поместим все те же объекты, перетащив их с навигатора модели, рис. Рисунок 3.65.

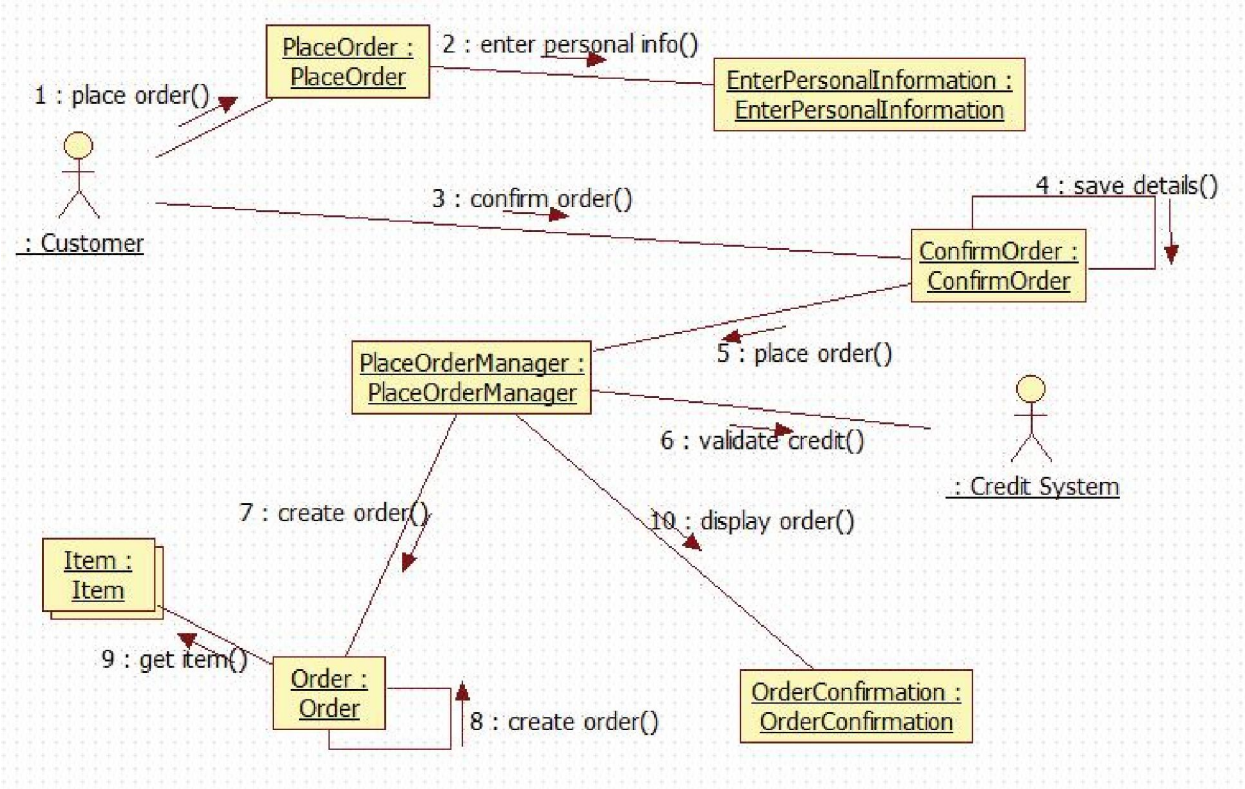


Рисунок 3.65 – Кооперативная диаграмма сценария Оформление заказа

3.4 Атрибуты, операции классов и диаграммы состояний

Механизм инкапсуляции в UML реализуется за счет объединения свойств и поведения в одном объекте. Свойства объекта описываются с помощью задания атрибутов класса, к которому относится объект, а поведение – заданием операций класса. На прямоугольнике класса атрибуты описываются во второй секции под именем, а операции – в третьей, под атрибутами. Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса. Атрибуты, таким образом, определяют структуру класса.

Атрибут (attribute) – содержательная характеристика класса, описывающая множество значений, которые могут принимать отдельные объекты этого класса.

Класс Товар (Item) может содержать следующие атрибуты: артикул, название, цена, размерный ряд, цвет, рис. 3.66.

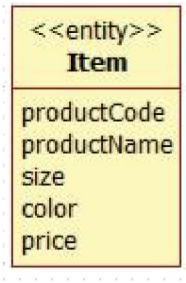


Рисунок 3.66 – Класс Товар с атрибутами

Для того чтобы создать атрибут класса в StarUML, надо выделить класс, атрибуты которого необходимо задать, щелкнув по нему один раз, затем в редакторе свойств Properties этого класса открыть раздел Attributes, нажав кнопку ... Откроется редактор коллекций (Collection Editor), содержащий вкладки редакторов атрибутов, операций и пр., рис. 3.67.

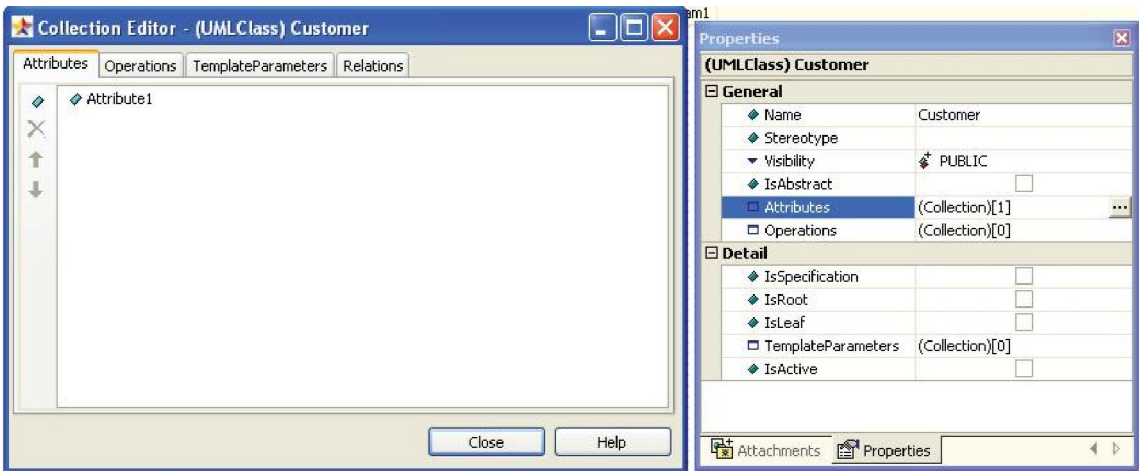


Рисунок 3.67 – Создание атрибута

Если в редакторе атрибутов нажать кнопку , то будет создан новый атрибут и откроется редактор его свойств, в котором можно изменить имя созданного атрибута (раздел Name) и задать другие спецификации (о спецификациях атрибутов и операций будет сказано ниже). Для сохранения атрибута и его спецификаций необходимо просто закрыть, нажав кнопку Close.

Создадим атрибуты классов сценария Оформление заказа. Для класса Заказ (Order) определим следующие атрибуты: номерЗаказа (code), имяПокупателя (firstName),

фамилияПокупателя (lastName), телефон (phoneNumber), электронныйАдрес (e-mail), товар (item), суммаЗаказа (total). Для класса Товар (Order) определим атрибуты артикул (productCode), название (productName), размер (size), цвет (color), цена (price). Диаграмма классов с атрибутами показа на рис. 3.68.

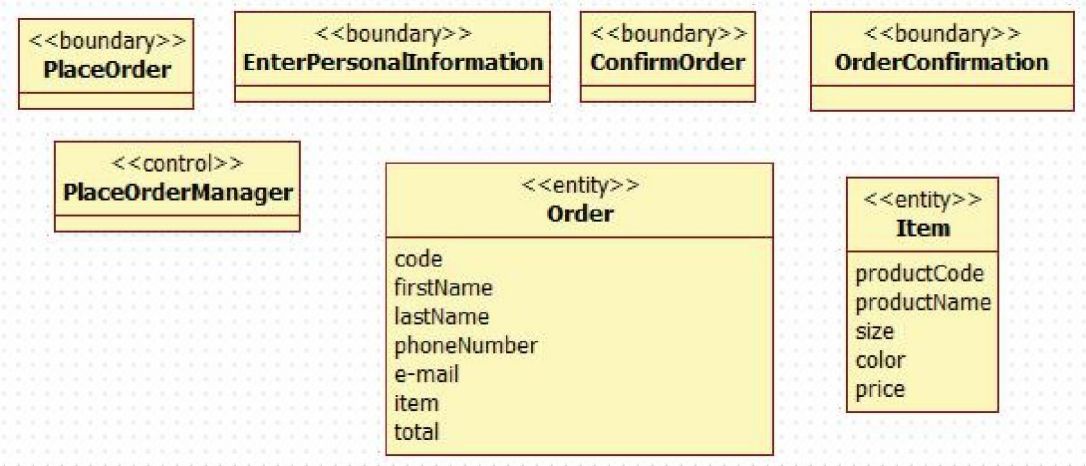


Рисунок 3.68 – Диаграмма классов сценария Оформление заказа с атрибутами классов

Операция (operation) – это сервис, предоставляемый каждым экземпляром или объектом класса по требованию своих клиентов, в качестве которых могут выступать другие объекты, в том числе и экземпляры данного класса.

Так, например класс ПодтверждениеЗаказа (OrderConfirmation) должен уметь отобразить информацию о заказе после его оформления, значит, он должен иметь соответствующую операцию: отобразитьЗаказ (displayOrder), рис. 3.69.

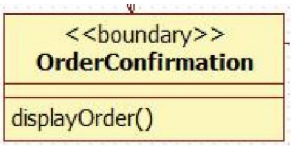


Рисунок 3.69 – Операция класса

Чтобы создать операцию класса в StarUML, щелкните один раз по этому классу, в редакторе свойств Properties откройте раздел Operations, нажав кнопку  . Откроется редактор коллекций (Collection Editor), в котором нужно перейти в редактор операций, выбрав вкладку Operations (Операции). Если в редакторе операций нажать кнопку  , то будет создана новая операция и откроется редактор ее свойств, в котором можно изменить имя созданной операции (раздел Name) и задать другие спецификации, рис. 3.70.

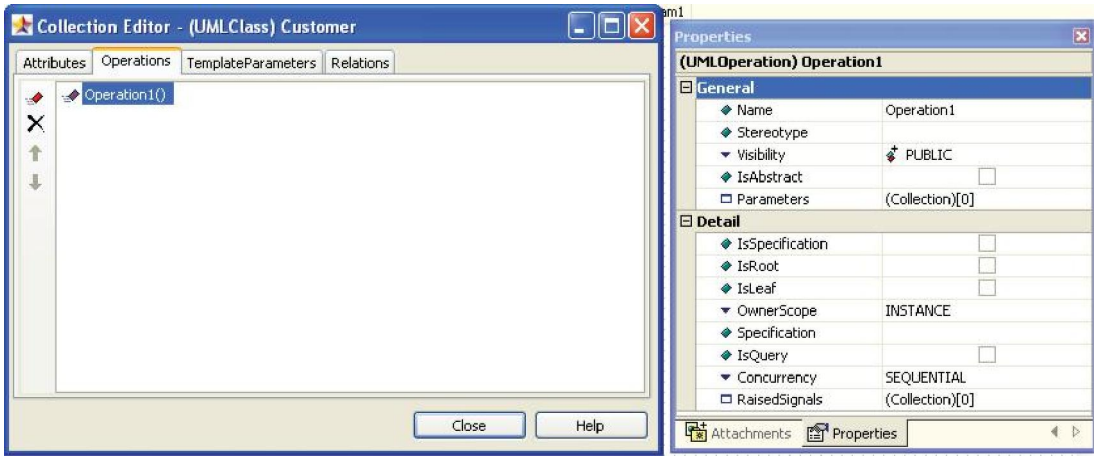


Рисунок 3.70 – Создание операции класса

Для сохранения операции и её спецификаций необходимо просто закрыть диалог, нажав кнопку Close.

Более быстрый способ создать атрибут или операцию – это щелкнуть два раза левой кнопкой мыши по классу, рис. 3.71. Нажав кнопку (для атрибута) или (для операции) будет создан атрибут или операция соответственно.

Рисунок 3.71 – Быстрое создание атрибута

Чтобы удалить атрибут или операцию необходимо щелкнуть по ней два раза левой кнопкой мыши и нажмите на значок справа. Чтобы добавить еще один атрибут или операцию надо нажать на значок , рис. 3.72.

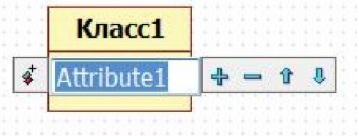


Рисунок 3.72 – Быстрое добавление и удаление атрибута

Сообщения, которые посылают объекты друг другу на диаграммах последовательности, как правило, вызывают определенную операцию класса этого объекта. Чтобы соотнести сообщение с операцией, нужно, чтобы каждому объекту был назначен классификатор, то есть класс, объектом которого он является.

Для того, чтобы соотнести объект с соответствующим ему классом, а классу присвоить объект, для чего есть, по крайней мере, два способа.

- Щелкнуть два раза по объекту, для которого нужно создать класс. Затем щелкнуть на значок справа от имени объекта и в появившееся окно ввести имя класса. Класс будет автоматически создан в Logical View, а имя объекта изменится на составное. Далее надо перетащить созданный класс на соответствующую диаграмму классов. (Объекту был назначен его классификатор.)
- Выделить объект, которому нужно назначить уже существующий классификатор. Справа на вкладке Properties (Свойства) выбрать раздел Classifier (Классификатор), нажать на значок и в появившемся диалоговом окне найти класс, соответствующий данному объекту.

После назначения классификаторов объектам диаграммы последовательности сценария прецедента Оформить заказ внешний вид диаграммы изменится, рис. 3.73.

Если объекты создавались на диаграмме последовательности с помощью перетаскивания классов с навигатора модели, то они автоматически связаны со своими классами.

В StarUML есть две возможности связать сообщение с операцией: можно создать операцию из сообщения, а можно использовать имя операции класса в качестве сообщения.

1 Для того чтобы создать операцию из сообщения, надо щелкнуть два раза по сообщению, нажать на справа от сообщения, и в открывшемся поле ввести имя новой операции.

2 Для того чтобы использовать операцию класса как сообщение, надо щелкнуть два раза по сообщению, нажать на слева от сообщения, и в появившемся списке выбрать нужную операцию.

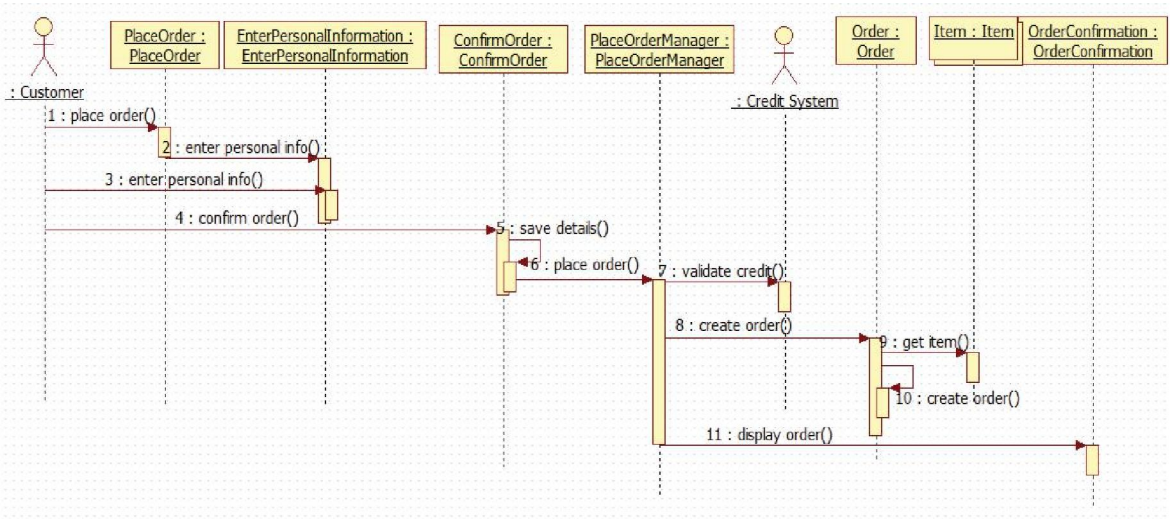


Рисунок 3.73 – Измененная диаграмма последовательности сценария Оформить заказ

Если подходить к понятию операции, как сервису, предоставляемому объектом другим объектам, то процедура создания операций из сообщений оказывается очень удобной для того, чтобы создавать операции классов.

На диаграмме последовательностей сценария Оформление заказа первое сообщение посылается объекту PlaceOrder. Этот объект должен уметь запускать оформление заказа, если корзина не пуста, создадим для него соответствующую операцию placeOrder.

Для класса EnterPersonalInformation необходимо создать соответствующую операцию enterPersonalInformation из сообщения, посылаемого объектом PlaceOrder, а не из такого сообщения, посылаемого Покупателем – для покупателя это сообщение означает заполнение полей формы. В класс OrderConfirmation необходимо добавим операцию displayOrder, создав её из 9-ого сообщения «display order».

Далее надо создать остальные операции классов и связать операции с сообщениями. В итоге на диаграмме последовательности вместо сообщений появятся операции, рис. 3.74.

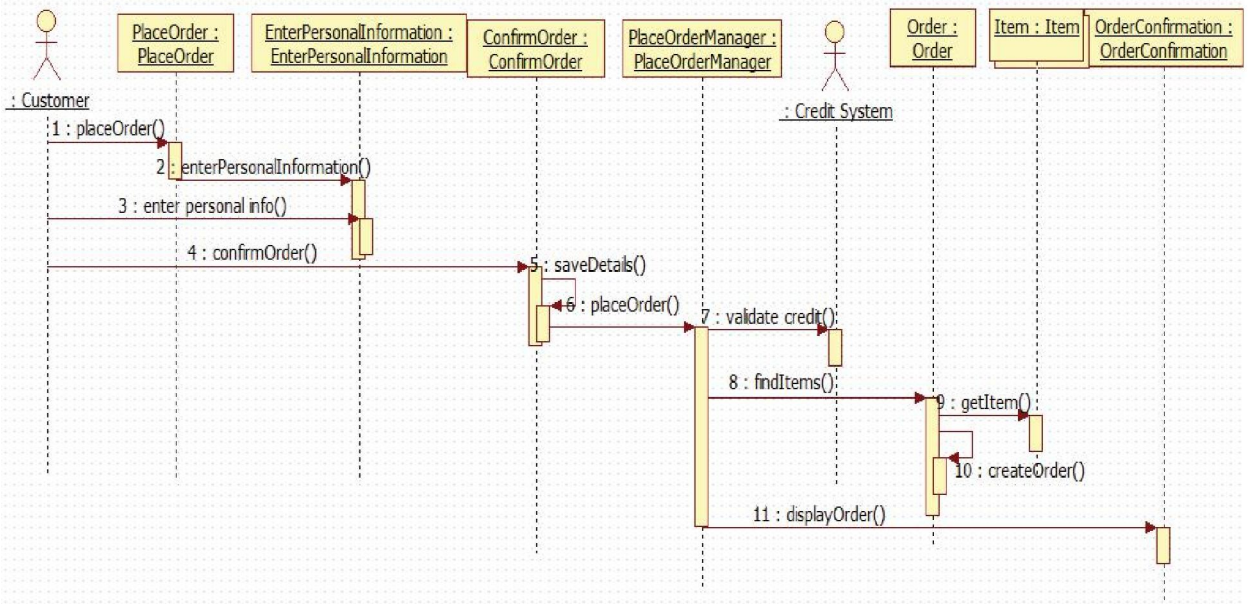


Рисунок 3.74 – Диаграмма последовательности с операциями

Для рассматриваемого случая диаграмма классов с операциями, атрибутами и стереотипами будет выглядеть так, как показано на рис. 3.75.

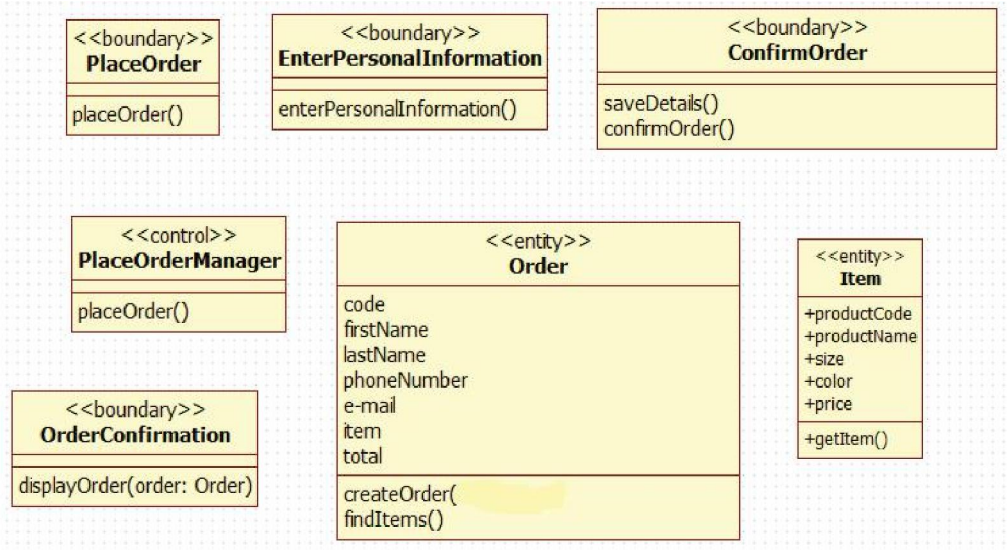


Рисунок 3.75 – Диаграмма классов с операциями

Язык UML позволяет специфицировать атрибуты и операции заданием их видимости, кратности и пр. Общий формат записи отдельного атрибута класса следующий:

[квантор видимости] имя атрибута [кратность] :

[тип атрибута] [= исходное значение] [{строка-свойство}]

Все элементы в квадратных скобках «[ ]» являются необязательными спецификациями атрибутов и могут быть опущены. Однако их использование позволяет сделать модель более полной и управлять взаимоотношениями между классами, разграничивая их права доступа.

Рассмотрим пример:

фамилия – указано только имя атрибута;

+фамилия – имя и видимость;

фамилия: String – имя и тип значений атрибута;

товаровВКорзине [0..*]: Integer – имя, кратность и тип;

-ID [1]: String {frozen} – видимость, имя, кратность, тип и свойство;

товаровВКорзине: Integer=0 – имя и начальное значение.

Опишем спецификации атрибутов подробно.

Имя атрибута может быть произвольной текстовой строкой. Имя является единственным обязательным элементом при задании атрибута. Имя должно начинаться с маленькой буквы, если оно содержит несколько слов, то остальные слова, кроме первого, пишутся с большой буквы:

Фамилия или *фамилияСотрудника*.

Видимость (visibility) – качественная характеристика описания свойств класса, характеризующая потенциальную возможность других объектов модели использовать это свойство (атрибут или операцию). Видимость в языке UML обозначается с помощью квантора видимости (visibility), который может принимать одно из 4-х возможных значений и отображаться при помощи специальных символов.

Открытый (public). Атрибут виден всем остальным классам. Любой класс, связанный с данным в рамках диаграммы или пакета, может просмотреть или изменить значение атрибута. Обозначается символом «+» перед именем атрибута.

Защищенный (protected). Любой потомок данного класса может пользоваться его защищенными свойствами. Обозначается знаком «#» перед именем атрибута.

Закрытый (private). Атрибут с этой областью видимости недоступен или не виден для всех классов без исключения. Обозначается знаком «-» перед именем атрибута.

Пакетный (package). Атрибут является открытым, но только в пределах своего пакета. В StarUML данный атрибут обозначается значком «~».

Когда создается атрибут класса, StarUML автоматически делает его открытым. Квантор видимости изображается рядом с именем атрибута, слева. Чтобы изменить видимость, надо выделить атрибут, щелкнув по нему два раза, далее щелкнуть по появившемуся значку  слева от имени атрибута и в списке выбрать желаемую видимость атрибута, рис. 3.76.

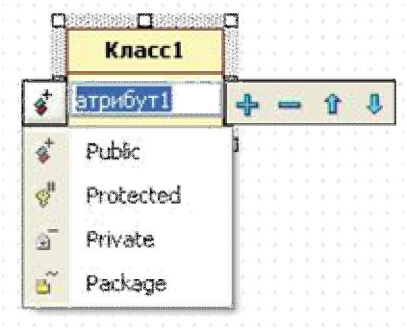


Рисунок 3.76 – Определение видимости атрибута

Изменить видимость атрибута можно, также используя редактор свойств данного атрибута, раздел Visibility (Видимость), рис. 3.77.

Рисунок 3.77 – Определение видимости атрибута из редактора свойств

В классе Item (Товар) атрибут productCode необходимо сделать защищенным. А в классе Заказ (Order) защищенным будет атрибут code, рис. 3.78.

Квантор видимости может быть опущен. Его отсутствие означает, что видимость атрибута не указывается. Вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private, package или использовать значок StarUML для обозначения видимости.

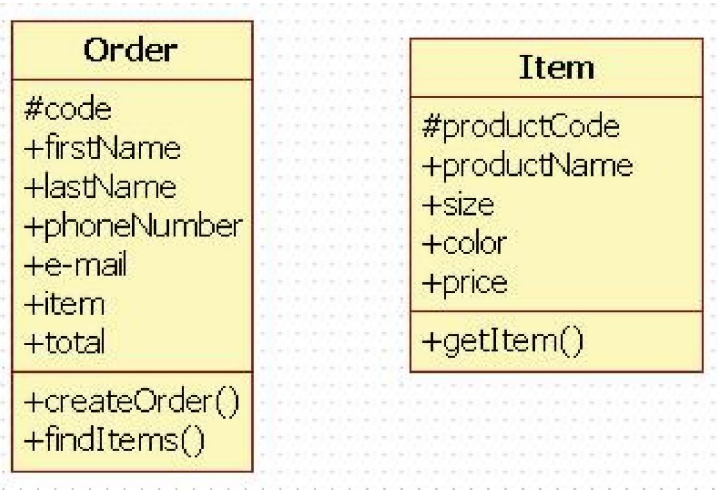


Рисунок 3.78 – Атрибуты классов Товар и Заказ с видимостью

Чтобы не отображать кванторы видимости на диаграмме, нужно выполнить следующие действия: щелкнуть правой кнопкой мыши по классу, в контекстном меню выбрать пункт Format, затем выбрать Show Compartment Vizibility, рис. 3.79.

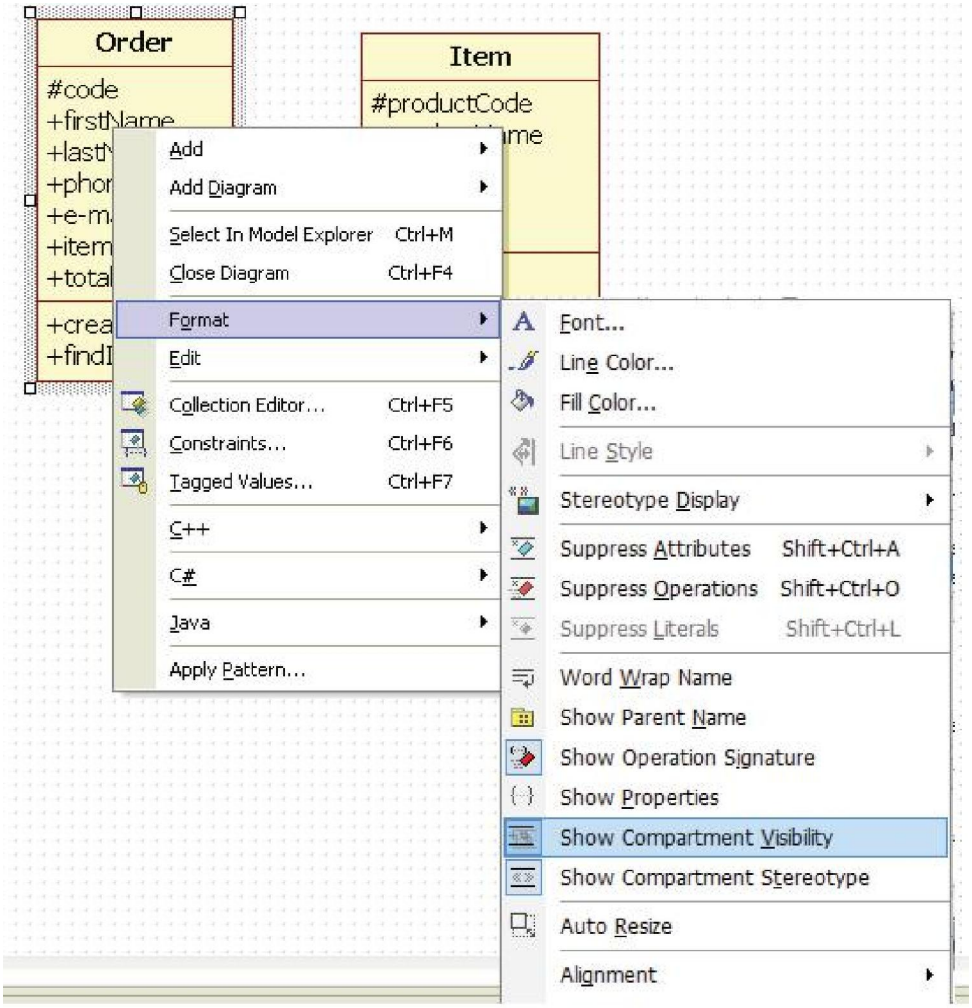


Рисунок 3.79 – Отображение кванторов видимости

Кратность (multiplicity) – спецификация области значений допустимой мощности, которой могут обладать соответствующие множества. Кратность указывает, сколько экземпляров данного атрибута может иметь экземпляр класса. Значение кратности записывается в квадратных скобках, в которых указывается возможный диапазон кратности атрибута: [нижняя граница .. верхняя граница], где нижняя и верхняя границы

положительные целые числа. В качестве верхней границы может использоваться специальный символ «*» (звездочка), который означает произвольное положительное целое число, т.е. неограниченное сверху значение кратности соответствующего атрибута.

Интервалов кратности отдельного атрибута может быть несколько. При этом придерживаются следующего правила: соответствующие нижние и верхние границы интервалов включаются в значение кратности. Если в качестве кратности указывается единственное число, то кратность атрибута принимается равной данному числу. Ниже приведены некоторые примеры записи кратности атрибута:

0..1	ноль или один
1 или 1..1	ровно один
2..*	два или больше
2..5	2, 3, 4 или 5
1..3, 5, 8..10	1, 2, 3, 5, 8, 9 или 10
*	любое положительное число или нуль

Для задания кратности атрибута в StarUML нужно найти атрибут, открыв раздел Attributes из редактора свойств соответствующего класса, выделить атрибут, после чего откроется его редактор свойств и в разделе Multiplisity надо выбрать кратность, рис. 3.80.

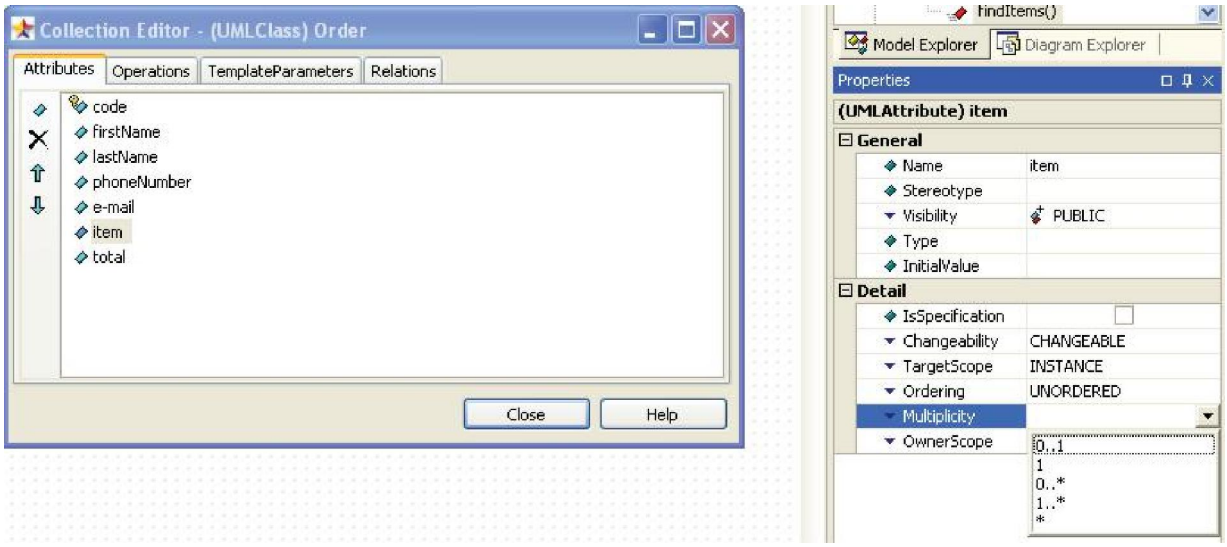


Рисунок 3.80 – Задание кратности атрибута

Как уже отмечалось ранее, каждый заказ должен содержать хотя бы один товар, но может состоять нескольких товаров. Чтобы позволить экземпляру класса Заказ (Order) иметь несколько экземпляров атрибута товар (item), необходимо определить кратность этого атрибута как [1..*], рис. 3.81.

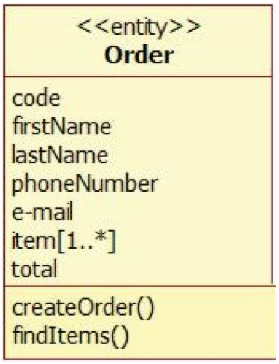


Рисунок 3.81 – Атрибут item с кратностью

Тип атрибута представляет собой выражение, семантика которого определяется некоторым типом данных, определенным в пакете «Типы данных языка UML» или самим разработчиком. Тип атрибута может задаваться в зависимости от языка программирования, который предполагается использовать для реализации данной модели. Если в качестве атрибута класса выступает другой класс, то типом атрибута будет этот класс. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс. Для задания типа атрибута в StarUML нужно найти атрибут, открыв раздел Attributes из редактора свойств соответствующего класса, выделить атрибут, и в открывшемся редакторе свойств, выбрать раздел Type, нажав ... В появившемся диалоговом окне выбрать один из стандартных типов, либо один из определенных пользователем типов (классов), рис. 3.82.

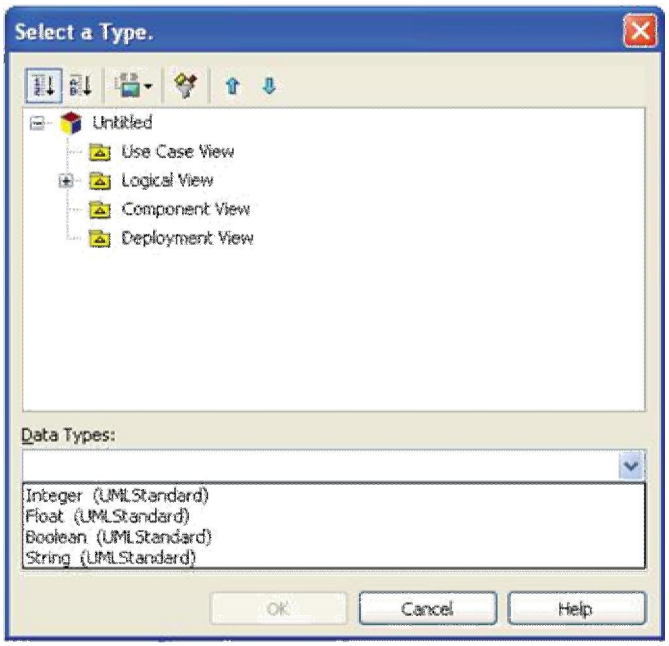


Рисунок 3.82 – Определение типа атрибута

Атрибут item класса Заказ (Order) должен быть экземпляром класса Товар (Item). Для остальных атрибутов используются стандартные типы, рис. 3.83.

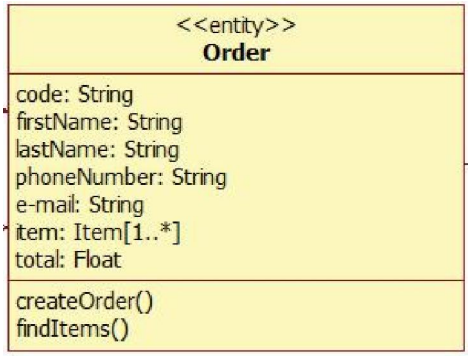


Рисунок 3.83 – Типы атрибутов класса Order

Исходное значение служит для задания начального значения соответствующего атрибута в момент создания отдельного экземпляра класса. Здесь необходимо придерживаться правила принадлежности значения типу конкретного атрибута. Если исходное значение не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса. С другой стороны, конструктор объекта

может переопределять исходное значение в процессе выполнения программы, если в этом возникает необходимость. Строка-свойства служит для указания дополнительных свойств атрибута, которые могут характеризовать особенности изменения значений атрибута в ходе выполнения программы. Фигурные скобки как раз и обозначают фиксированное значение соответствующего атрибута для класса в целом, которое должны принимать все вновь создаваемые экземпляры класса без исключения. Это значение принимается за исходное значение атрибута, которое не может быть переопределено в последующем. Отсутствие строки-свойства по умолчанию трактуется так, что значение соответствующего атрибута может быть изменено в программе.

Поведенческие характеристики класса моделируются определением операций класса. На первом уровне абстракции достаточно просто записать их имена, но для операций, также как и для атрибутов, определен ряд спецификаций. Наиболее полный синтаксис записи операции в UML следующий:

[квантор видимости] имя операции [(список параметров)] [: выражение типа возвращаемого значения] [{строка-свойство}]

Все элементы, стоящие в квадратных скобках «[]», являются необязательными спецификациями операций, однако наличие круглых скобок в описании операции обязательно, даже если список параметров пуст.

Рассмотрим пример:

отобразить() – указано только имя операции;

+отобразить() – имя и видимость;

+добавитьТоварВКорзину (inout t:Товар) – видимость, имя, параметр, тип параметра и его направление;

удалитьТоварИзКорзины (q:ТоварВКорзине) – указаны имя и параметр;

изменитьКоличествоТовара (q:Товар в корзине, inout n: Integer): Integer – имя операции, параметры, направление параметра и тип возвращаемого значения.

Раскроем смысл спецификаций операций.

Имя операции совместно с ее параметрами называют сигнатурой операции. Имя операции – это строка текста, обычно используют глагол или короткое глагольное выражение, если оно состоит из нескольких слов, то все слова, кроме первого, пишутся с большой буквы: добавить или добавитьТоварВКорзину.

Так же как и для атрибутов класса, видимость (visibility) операции обозначается с помощью квантора видимости и имеет четыре допустимых значения:

Открытый (public). Атрибут виден всем остальным классам. Любой класс, связанный с данным в рамках диаграммы или пакета, может просмотреть или изменить значение атрибута. Обозначается символом «+» перед именем атрибута.

Защищенный (protected). Любой потомок данного класса может пользоваться его защищенными свойствами. Обозначается знаком «#» перед именем атрибута.

Закрытый (private). Атрибут с этой областью видимости недоступен или не виден для всех классов без исключения. Обозначается знаком «-» перед именем атрибута.

Пакетный (package). Атрибут является открытым, но только в пределах своего пакета. В StarUML данный атрибут обозначается значком «~».

Квантор видимости для операции может быть опущен. В этом случае его отсутствие просто означает, что видимость операции не указывается. Вместо условных графических

обозначений также можно записывать соответствующее ключевое слово: public, protected, private, package.

Квантор видимости атрибутов и операций также может быть указан в виде специального значка или символа, которые используются для графического представления моделей в инструментальном средстве.

Для класса Товар (Item) ранее определена операция getItem(), которая должна быть доступна классу Заказ (Order) для вызова, остальным классам знать об этой операции не обязательно. Поэтому её надо сделать защищенной, рис. 3.84.

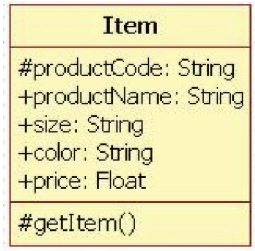


Рисунок 3.84 – Защищенная операция класса

Сигнатура операции может содержать ноль или более параметров, каждый из которых имеет следующий синтаксис: [направление параметра] имя параметра: тип параметра [= значение параметра по умолчанию]. Имя параметра есть идентификатор соответствующего формального параметра, при записи которого следуют правилам задания имен. Тип параметра является спецификацией типа данных для допустимых значений соответствующего формального параметра. Тип данных параметра может быть стандартным типом UML, либо, если в качестве входного параметра операция использует целый класс, то параметр имеет типом этот класс. В StarUML допускаются четыре стандартных типа UML: String, Integer, Float, Boolean. Параметр может также быть типом класса, который используется в данной операции. Тип параметра записывается с большой буквы.

Для задания параметра операции в StarUML нужно найти атрибут, открыв раздел Operations из редактора свойств соответствующего класса, выделить операцию, откроется её редактор свойств, в котором надо открыть раздел Parameters, нажав  . В появившемся диалоговом окне, чтобы создать параметр, требуется нажать значок  . Параметр будет создан со стандартным именем Parameter1. Для удаления параметра используется кнопка  . При создании параметра откроется его редактор свойств, в котором можно изменить имя параметра и определить его тип, открыв раздел Type, рис. 3.85.

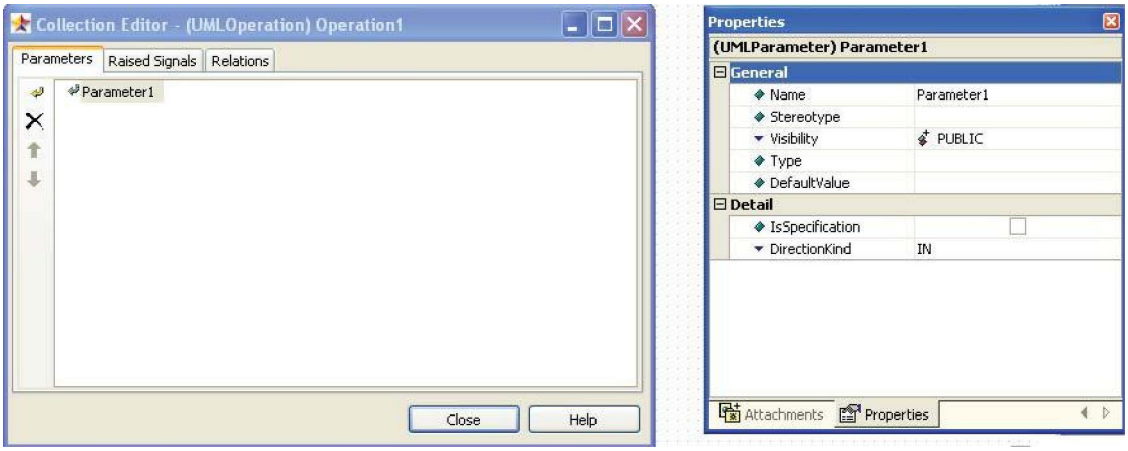


Рисунок 3.85 – Добавление параметра операции

В рассматриваемом примере Оформление заказа, чтобы создать подтверждение заказа и показать его на экране, классу OrderConfirmation (ПодтверждениеЗаказа) нужна информация о заказе. Для того чтобы OrderConfirmation (ПодтверждениеЗаказа) узнавал об изменениях в классе Order (Заказ) нужно в OrderConfirmation (ПодтверждениеЗаказа) использовать Order (Заказ) в качестве параметра операции, рис. 3.86.

Рисунок 3.86 – Параметр операции типа Order

Направление параметра может принимать одно из нижеследующих значений:

in – входящий параметр, который не может быть изменен;

out – выходящий параметр, который может быть изменен, чтобы передать информацию вызвавшей процедуре;

inout – входящий параметр, который может быть изменен.

По умолчанию, если направление параметра не указано, принимается значение in. Значение параметра по умолчанию в общем случае представляет собой некоторое конкретное значение для этого формального параметра.

Выражение типа возвращаемого значения операции также указывает на тип данных значения, которое возвращается объектом после выполнения соответствующей операции. Оно может быть опущено, если операция не возвращает никакого значения. Для указания нескольких возвращаемых значений данный элемент спецификации операции может быть записан в виде списка отдельных выражений. Выражение типа возвращаемого значения еще называют возвращаемым классом операции. Для определения возвращаемого класса можно использовать встроенные типы (string, float, integer, boolean) или типы, определенные в модели.

Для того чтобы класс Order (Заказ) мог найти товары и добавить их в заказ, ранее была определена операция findItem(). Результатом выполнения этой операции может быть элемент типа item, рис. 3.87.

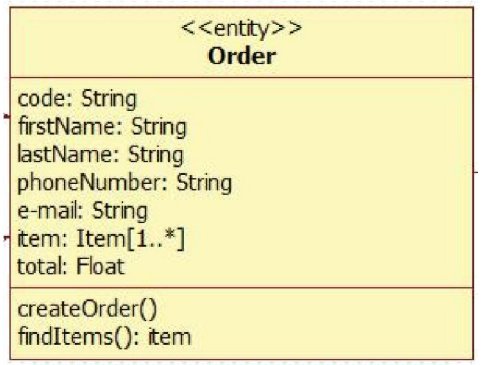


Рисунок 3.87 – Возвращаемое значение типа item

Строка-свойство в описании операции служит для указания значений свойств, которые могут быть применены к данной операции. Строка-свойство может отсутствовать, если свойства не специфицированы.

Отношение – это связь между классами. Для того чтобы обнаружить связи классов исследуются сценарии и диаграммы последовательности: если объект посылает сообщение другому объекту, то по-видимому, между ними существует отношение.

Отношения возникают также, если один класс использует другой в качестве параметра операции.

В нотации UML определены 5 видов отношений между классами:

- ассоциации;
- агрегации;
- композиции;
- зависимости;
- обобщения.

Отношение между классами изображается на диаграммах стрелками, вид стрелки зависит от семантики отношения. Чтобы создать отношение между классами в StarUML надо щелкнуть один раз по обозначению элемента связи на панели элементов слева, выбрать тот тип отношения, который необходим, а затем провести линию от одного класса к другому, удерживая левую кнопку мыши, рис. 3.88.

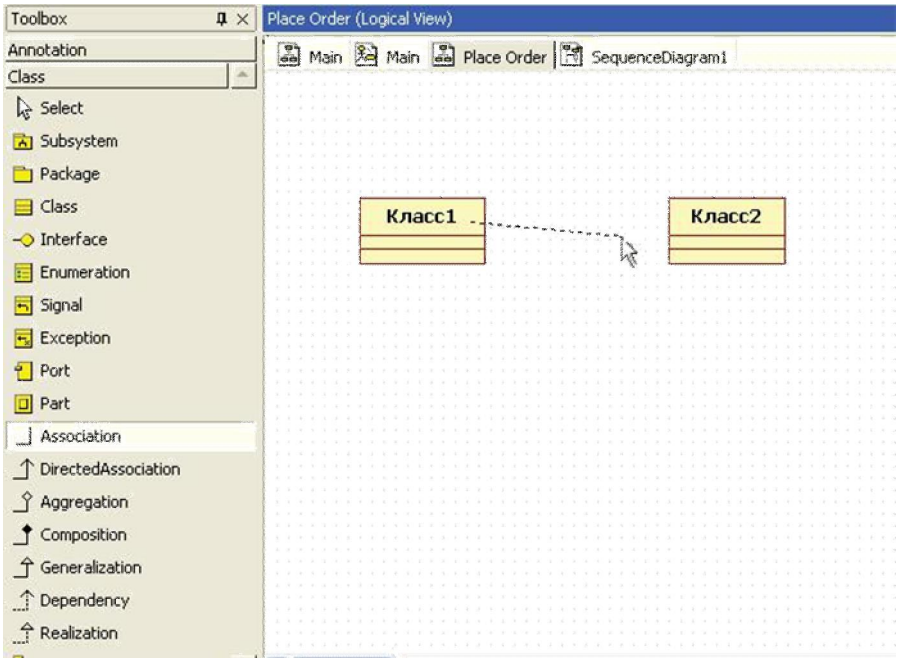


Рисунок 3.88 – Создание отношения между классами

Ассоциация – это семантическая связь между классами. На диаграмме ее рисуют в виде обычной линии. Ассоциация может быть однонаправленной или двунаправленной. В первом случае ее изображают в виде стрелки, показывающей направление связи. Во втором случае – двойной стрелки либо просто линии без стрелок. Если между классами создана двунаправленная связь, то каждый из них видит открытые атрибуты и операции других классов, рис. 3.89.

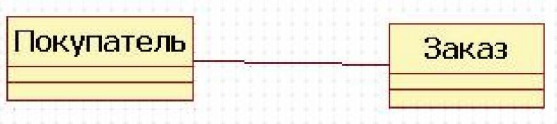


Рисунок 3.89 – Двунаправленная ассоциация

Если между классами установлена однонаправленная ассоциация, то в этом случае класс, от которого направлена стрелка, знает открытые атрибуты и операции второго класса, а второй класс, к которому идет стрелка, не видит атрибуты и операции первого класса. Например, на рис. 3.90 класс Покупатель знает название и размерный ряд Товара, но Товар не знает имя и адрес Покупателя.

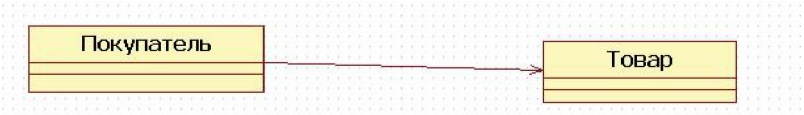


Рисунок 3.90 – Направленная ассоциация

При таком отношении между классами Покупатель должен знать о классе Товар и потому не может использоваться без него, но класс Товар не зависит от Покупателя и может быть повторно использован самостоятельно, независимо от класса Покупатель. Ассоциация может быть рефлексивной, это означает, что один экземпляр класса обращается к другому экземпляру этого класса. Изображается рефлексивное отношение как на рис. 3.91.

Рисунок 3.91 – Рефлексивная ассоциация

К ассоциации применимы четыре базовых дополнения: имя, роль, кратность и агрегирование. Ассоциации можно присвоить имя. Обычно в качестве имени используется глагол или предложение, поясняющее причину возникновения связи, рис. 3.92.

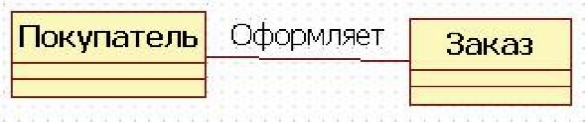


Рисунок 3.92 – Имя ассоциации

Окончание линии ассоциации в месте, где она соединяется с классом, называется ролью ассоциации. Название роли ассоциации – это обычно существительное, которое уточняет, описывает роль, в которой один класс участвует в связи с другим классом. Роль может быть указана для обоих классов, участвующих в связи, либо только для одного (рис. 3.93).

Рисунок 3.93 – Ассоциация с ролями

Кратность (мощность) определяется для классов и указывает допустимое количество объектов (экземпляров класса), участвующих в отношении. Кратность указывает, сколько экземпляров одного класса взаимодействуют с помощью отношения с одним экземпляром данного класса в данный момент.

Примеры индикаторов:

0..1	ноль или один;	
1	ровно один;	
1..*	один или много;	
2..5	2, 3, 4	или 5
6..8, 10	6, 7, 8	или 10

Покупатель может оформить много заказов или не оформить ни одного. Заказ должен быть сделан только 1 покупателем, рис. 3.94.

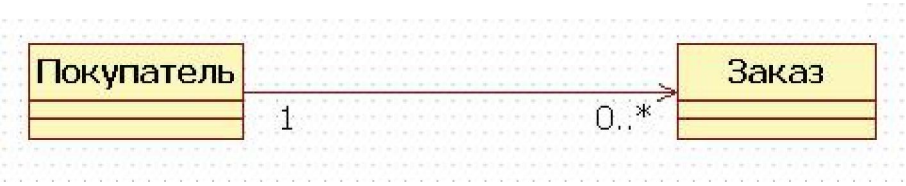


Рисунок 3.94 - Кратность классов в ассоциации

Как видно из примера, читать кратность класса нужно на противоположном конце связи. Указывать имя, роль или кратность связи необязательно. Это нужно делать, когда это может помочь более точному представлению модели системы и лучшему ее пониманию.

Агрегация – специальная форма ассоциации, которая служит для представления отношения типа «часть-целое» между агрегатом (целое) и его составной частью, рис. 3.95.

Рисунок 3.95 - Агрегация

Графически агрегация изображается линией, на одном конце которой, принадлежащем целому, помещен не закрашенный ромб.

Любой заказ состоит из товаров, рис. 3.96.

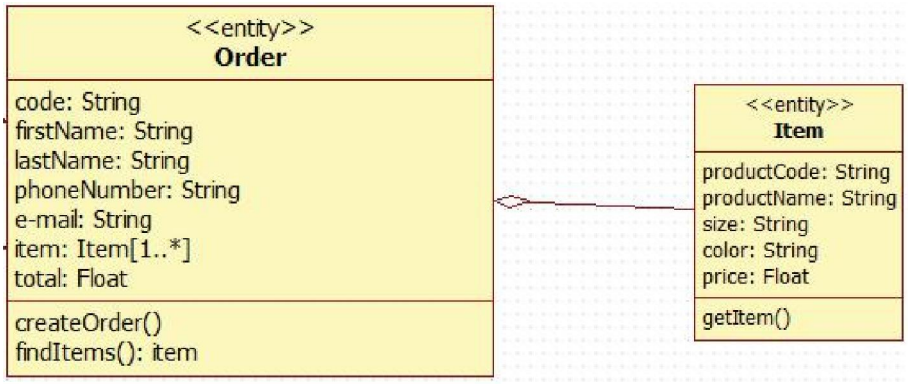


Рисунок 3.96 - Агрегация между классами Товар и Заказ

Как и ассоциации, агрегации могут быть рефлексивными. Это означает, что один экземпляр класса состоит из одного или нескольких экземпляров того же класса. Композицией называется форма агрегирования с четко выраженным отношением владения, причем время жизни частей и целого совпадают.

Как только будет уничтожен объект Целое, так вместе с ним будет уничтожен объект Часть. На диаграммах композиция показывается так же, как и агрегация, но только ромб должен быть закрашен. Классический пример: если открыто окно программы, то пользователь видит доступные ему кнопки, например, кнопку «Добавить» (товар). Как только окно будет закрыто, кнопки перестанут существовать. Это продемонстрировано на рис. 3.97. Так как агрегация и композиция являются отношениями ассоциации, то для них допустимо, но не обязательно, указывать имя, роль и кратность.

Рисунок 3.97 - Композиция

Зависимостью называется отношение использования, определяющее, что изменение в спецификации одной сущности может повлиять на другую сущность, которая ее использует, причем обратное в общем случае не верно. Графически зависимость изображается в виде пунктирной стрелки, которая идет к той сущности, от которой зависит еще одна. Зависимости применяются, чтобы показать, что один класс использует другой. Т.е. один класс является клиентом другого класса-поставщика и использует этот класс-поставщик как параметр своей операции.

В рассматриваемом примере используется класс Order (Заказ) как входной параметр операции displayOrder (отобразитьЗаказ) класса OrderConfirmation (ПодтверждениеЗаказа). Так как для выполнения этой операции, класс OrderConfirmation (ПодтверждениеЗаказа) использует класс Order (Заказ), то они связаны отношением зависимости, что продемонстрировано на рис. 3.98. Создание экземпляра класса ПодтверждениеЗаказа не повлечет за собой создание экземпляра класса Заказ. Однако эти два класса смогут обмениваться сообщениями на диаграммах взаимодействия.

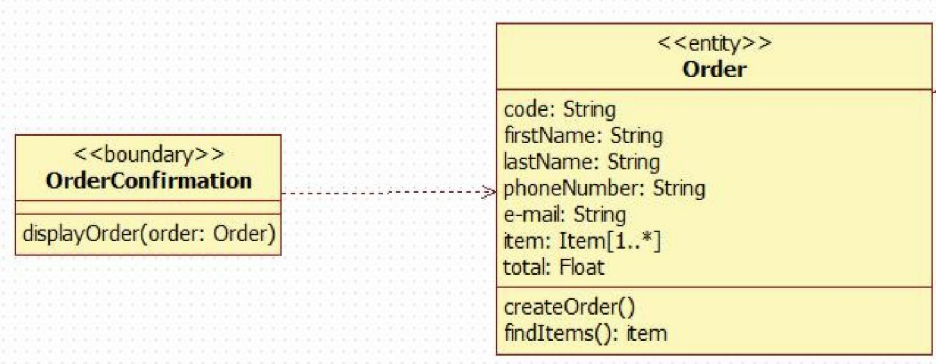


Рисунок 3.98 - Зависимость между классами

Обобщение – это отношение наследования между двумя элементами модели. Оно дает классу возможность наследовать открытые или защищенные атрибуты и операции суперкласса (класса от которого наследуются атрибуты и операции). Помимо наследуемых, каждый класс может иметь свои атрибуты и операции. На диаграммах обобщение изображается в виде стрелки с не закрашенным треугольником у суперкласса, идущей от потомка.

Так, в магазине могут работать различные сотрудники: сотрудник отдела продаж, кладовщик, директор. Все они имеют общие свойства: имя, адрес, телефон, дата рождения, должность, поэтому можно рассматривать обобщающую сущность Сотрудник, атрибуты и операции которой сущности Директор и Кладовщик будут наследовать, рис. 3.99. Если Сотрудник имеет в качестве атрибутов имя, адрес, телефон, дату рождения, должность, то сущности, Директор и Кладовщик, конечно, наследуют эти

атрибуты со своими значениями. Кроме того они могут иметь и собственные атрибуты или операции. Например, у Директора может быть операция уволитьСотрудника, которой не может быть у Кладовщика, а у последнего операция – выдатьТовар. Стоит помнить, что закрытые атрибуты и операции не могут наследоваться потомками.

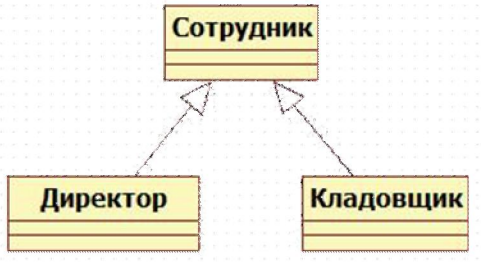


Рисунок 3.99 - Отношение обобщения между классами

Определим отношения между классами сценария Оформление заказа, рис. 3.100. Проанализировав диаграмму последовательности видно, что класс PlaceOrder связан с EnterPersonalInformation, а объект ConfirmOrder посылает сообщения объекту класса PlaceOrderManager. Последний, в свою очередь, связан с объектами классов Order и OrderConfirmation. Для всех перечисленных связей определим отношения ассоциации.

Класс OrderConfirmation использует класс Order как параметр своей операции: между ними требуется отношение зависимости. Экземпляры класса Order состоят из экземпляров класса Item. Между ними устанавливается отношение агрегации. Для того чтобы класс ConfirmOrder мог выполнять операцию подтверждения заказа, он должен быть связан с классом EnterPersonalInformation, поэтому между ними требуется отношение ассоциации. Класс OrderConfirmation использует класс Order как параметр своей операции: между ними устанавливается отношение зависимости. Экземпляры класса Order состоят из экземпляров класса Item, тогда между ними устанавливается отношение агрегации. Для того чтобы класс ConfirmOrder мог выполнять операцию подтверждения заказа, он должен быть связан с классом EnterPersonalInformation, поэтому между ними требуется отношение ассоциации.

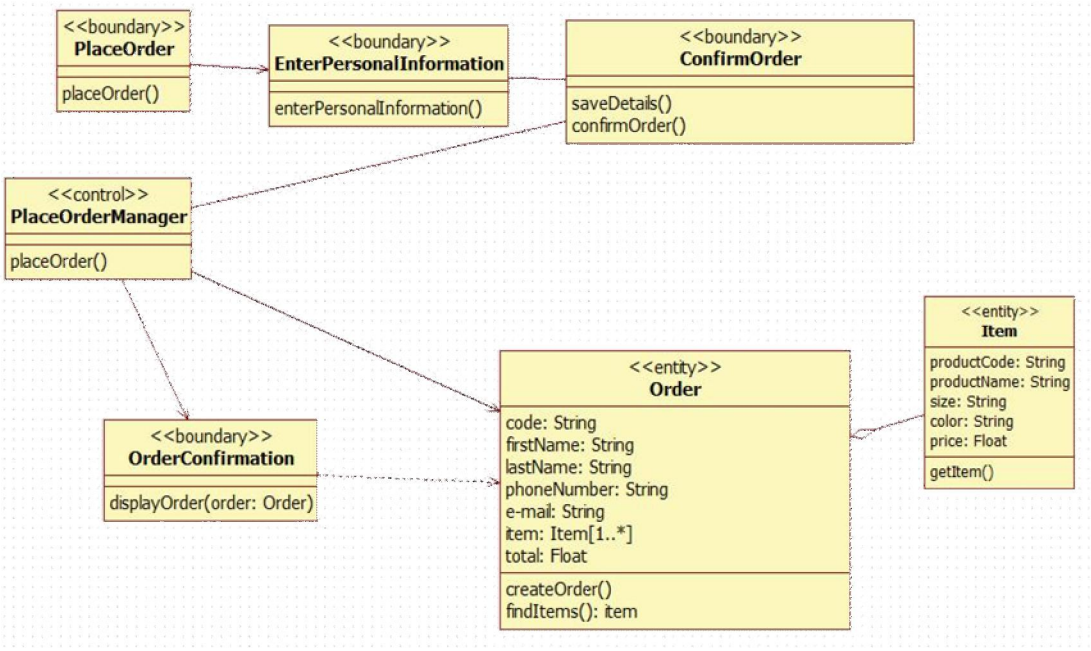


Рисунок 3.100 - Диаграмма классов с отношениями

Отношения между пакетами могут быть только отношением типа зависимости. Обозначается такое отношение пунктирной стрелкой, как показано на рис. 3.101. Если Пакет1 зависит от Пакета2, как показано на рисунке, то это значит, что один или несколько классов Пакета1 инициируют связь с общедоступными классами Пакета2. Пакет1 называется пакетом-клиентом, Пакет2 называется пакетом-поставщиком. Метод создания связей между пакетами в StarUML такой же, как и между классами.

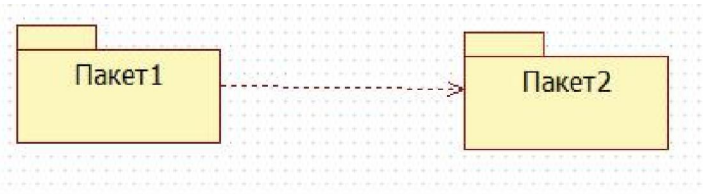


Рисунок 3.101 - Зависимость между пакетами

Для представления взаимодействий объектов в системе используют диаграммы последовательности и кооперации. Прецеденты и сценарии описывают поведение системы, но если необходима 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.

Для добавления диаграммы состояний в модель нужно выполнить следующие шаги: щелкнуть правой кнопкой мыши по папке представления Logical View в навигаторе модели, в контекстном меню выбрать пункт Add Diagram, и в списке выбрать диаграмму состояний Statechart Diagram, рис. 3.102. Также можно связать диаграмму состояний с тем классом, состояния объекта которого она описывает. Для этого нужно щелкнуть правой мышкой по соответствующему классу, а не по папке Logical View.

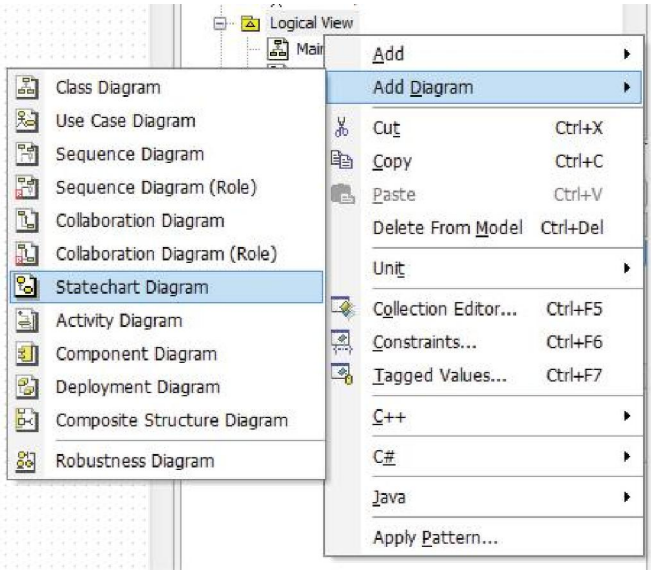


Рисунок 3.102 - Добавление диаграммы состояний

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

Автомат (State machine) – 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. 3.103.

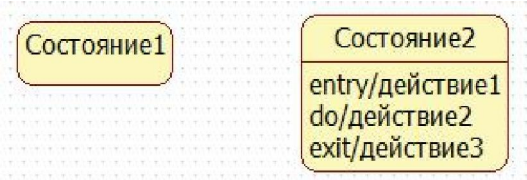


Рисунок 3.103 - Состояния

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 (start state) – 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. Конечное состояние (final state) – разновидность псевдосостояния, обозначающее прекращение процесса изменения состояний конечного автомата. Графическое представление состояний приведено на рис. 3.104. Объект находится в конечном состоянии непосредственно перед уничтожением. Конечных состояний может быть несколько.

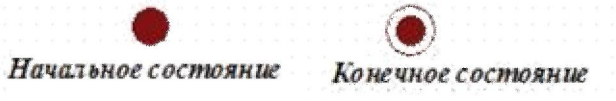


Рисунок 3.104 - Начальное и конечное состояния

В рассматриваемом примере Покупатель оформляет заказ. Класс Заказ, кроме прочих атрибутов имеет атрибут «статус». Проследим динамику движения заказов в системе с помощью диаграммы состояний, составленной для класса Заказ. По условию задачи данные о сделанном заказе поступают сотруднику отдела продаж, который проверяет оплату, реквизиты заказа и передает его кладовщику на комплектацию. Кладовщик, проверив наличие заказанных товаров и собрав заказ, если это возможно, делает отметку о готовности. Заказ выдается со склада кладовщиком. Кладовщик выдает

заказ и отмечает в системе, что заказ выдан. Далее данные о заказе могут быть переданы в архив. На рис. 3.105 показан переход заказа между состояниями.

Рисунок 3.105 Диаграмма состояний объекта класса Заказ

Находясь в каком-либо состоянии, объект может выполнять определенные действия.

Действие (Action) – это одиночное вычисление, которое приводит к смене состояния или возврату значения. В общем случае действие имеет следующий формат: <метка действия / выражение действия>

Входное действие (entry action) – действие, которое выполняется в момент перехода в данное состояние. Обозначается с помощью ключевого слова-метки действия entry, которое указывает на то, что следующее за ней выражение действия должно быть выполнено в момент входа в данное состояние.

Действие выхода (exit action) – действие, производимое при выходе из данного состояния. Обозначается с помощью ключевого слова-метки действия exit, которое указывает на то, что следующее за ней выражение действия должно быть выполнено в момент выхода из данного состояния.

Внутренняя деятельность (do activity) – выполнение объектом операций или процедур, которые требуют определенного времени. Другими словами, деятельность – это поведение, которое реализуется объектом, когда он находится в данном состоянии. Деятельность – это прерываемое действие: оно может выполняться до конца или быть прервано переходом в другое состояние. Обозначается с помощью ключевого слова-метки деятельности do, которое специфицирует так называемую «ду-деятельность», выполняемую в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не будет прервано внешним событием. При нормальном завершении внутренней деятельности генерируется соответствующее событие.

В рассматриваемом примере при оформлении заказа он должен быть оплачен (входное действие Оплатить заказ), обработка заказа подразумевает проверку оплаты и наличия товаров (деятельность Проверить оплату и наличие), переход в одно из состояний На комплектации, Укомплектован, Выдан означает смену статуса заказа (соответствует действию выхода Изменить статус), рис. 3.106.

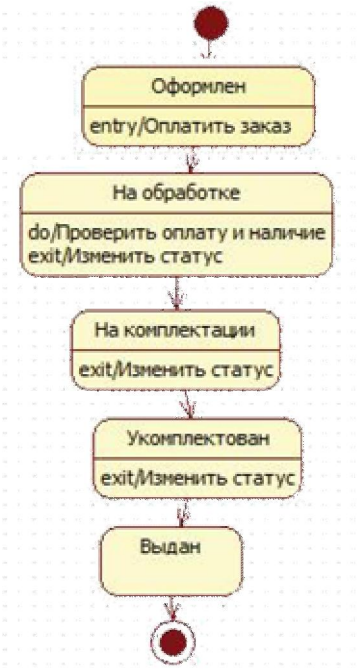


Рисунок 3.106 - Диаграмма состояний с деятельностями

Изменение состояния объекта осуществляется с помощью переходов. Переход (Transition) показывает, что объект, находящийся в некотором состоянии, должен выполнить некоторые действия и перейти в другое состояние, когда произойдет определенное событие, и будут выполнены соответствующие условия. Переход может быть направлен в то же состояние, из которого он выходит. В этом случае его называют переходом в себя. Исходное и целевое состояния перехода в себя совпадают. Этот переход изображается петлей со стрелкой и отличается от внутреннего перехода. При переходе в себя объект покидает исходное состояние, а затем снова входит в него. При этом всякий раз выполняются внутренние действия, специфицированные метками entry и exit.

Срабатывание <перехода> (fire) –- выполнение перехода из одного состояния в другое. На диаграмме переход изображается сплошной стрелкой. У перехода существует несколько спецификаций: событие, граничные 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. У события могут быть аргументы, которые записываются в скобках. Граничные (ограждающие) условия определяют, когда может быть выполнен переход, а когда – нет. Условия записываются в квадратных скобках. После условий может указываться действие – непрерываемое поведение, выполняемое как часть перехода.

В рассматриваемом примере если покупатель получил заказ, то это событие вызывает переход из состояния Укомплектован в состояние Выдан. Если же покупатель не получил свой заказ в течение двух недель, то заказ расформировывается, а деньги возвращаются покупателю на банковскую карту. Условие [Покупатель не забрал заказ в течение 2 недель] вызывает переход в состояние Расформирован при этом выполняется действие Вернуть деньги на карту. Окончательную диаграмму состояний можно видеть на рис. 3.107.

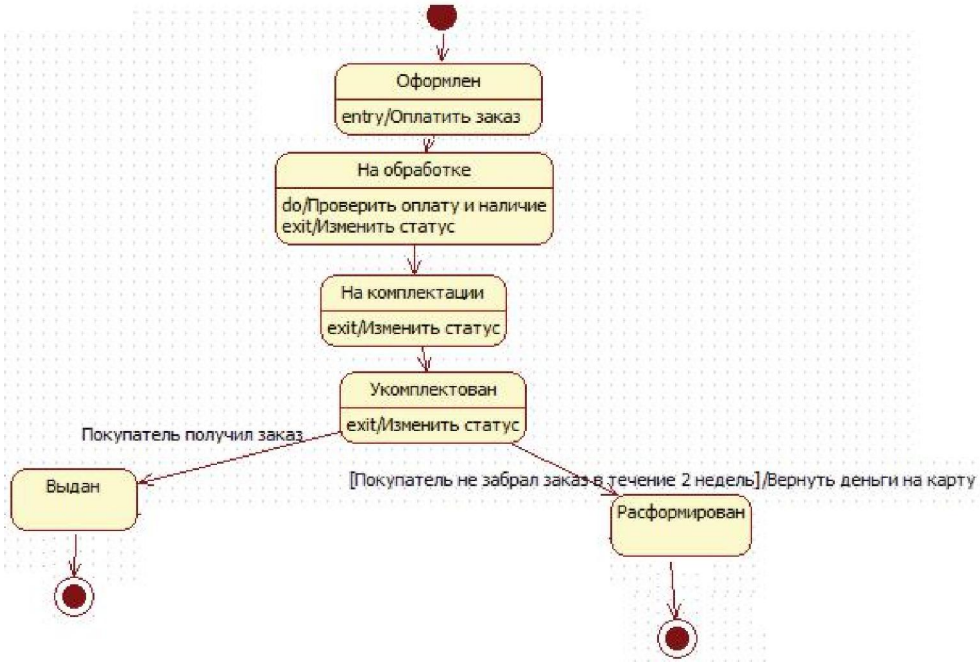


Рисунок 3.107 - Окончательная диаграмма состояний объекта Заказ

3.5 Итоговые диаграммы проекта моделирования системы заказов магазина «Style»

Итак, в ходе работы над проектом моделирования системы заказов магазина «Style» были определены прецеденты, актеры, классы и объекты системы и отношения между ними, описан поток событий основного прецедента, а также построены диаграммы прецедентов, деятельности, классов, пакетов, взаимодействия (последовательности и кооперации), состояний, итоговый вид которых показан на рисунках ниже.

Для системы заказов магазина «Style» были определены актеры Покупатель, Сотрудник, Система Склад и прецеденты Заказ товаров, Управление статусом заказа, Получение информации о заказе, рис. 3.108.

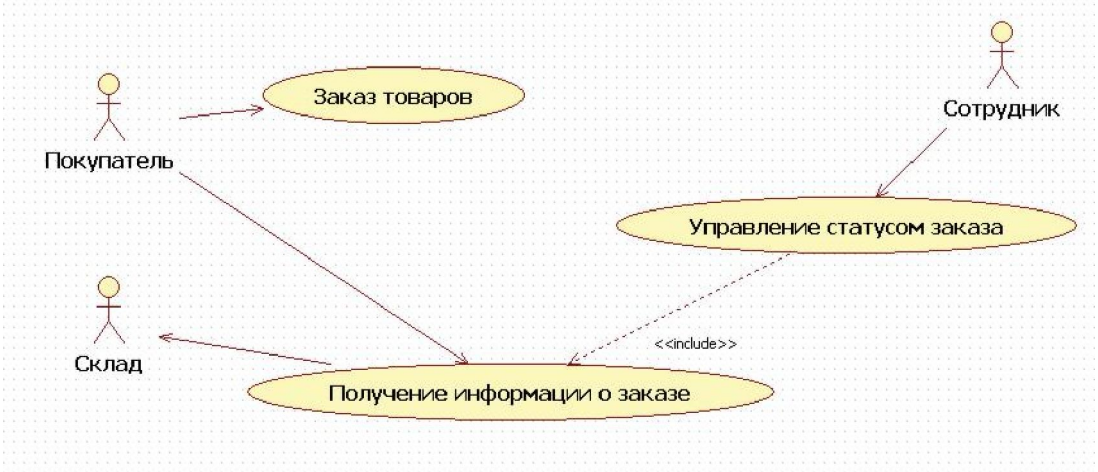


Рисунок 3.108 - Основная диаграмма прецедентов

Для прецедента Заказ товаров построена дополнительная диаграмма вариантов использования, рис. 3.109.

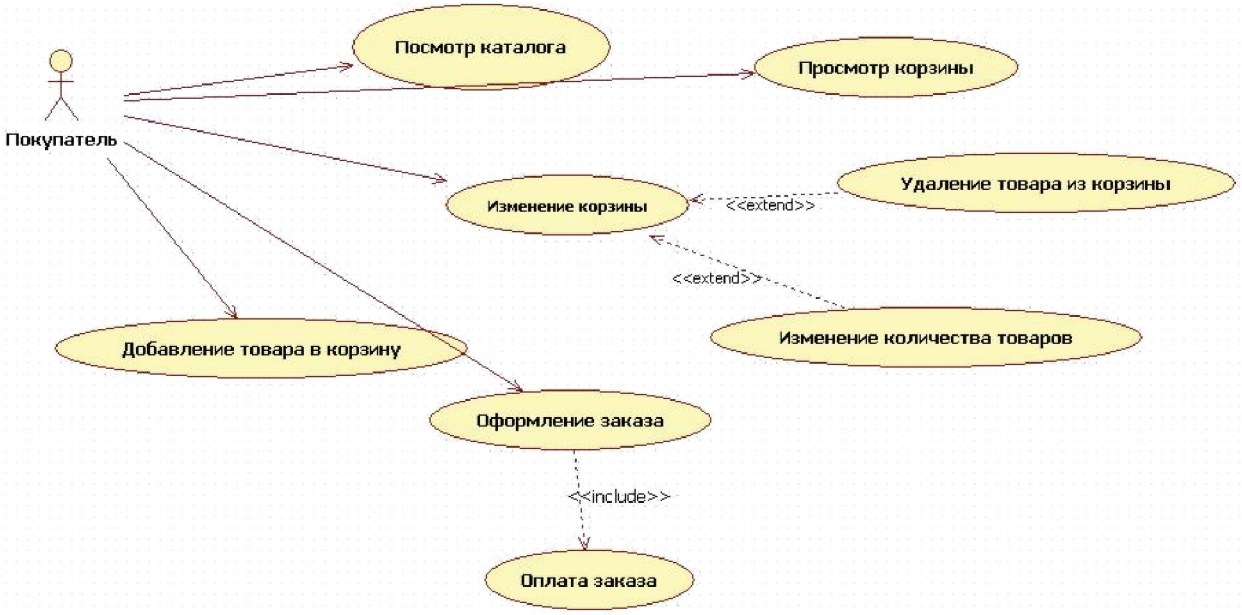


Рисунок 3.109 - Дополнительная диаграмма прецедентов

Часто в проекте присутствует не одна диаграмма классов каждого сценария, а несколько: на одной изображают только классы (рис. 3.110), на других – классы с атрибутами и операциями (рис. 3.111) и отношениями (рис. 3.112).

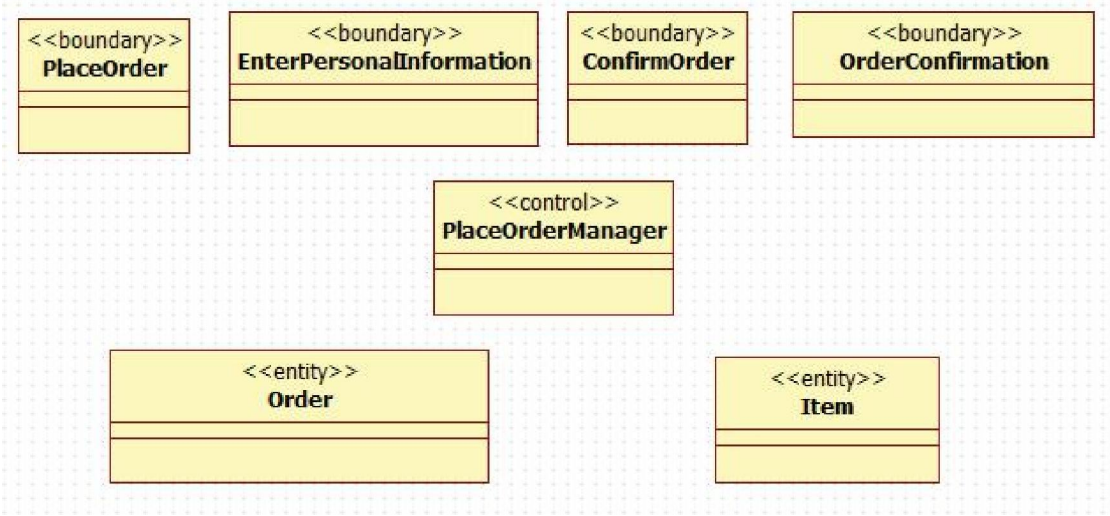


Рисунок 3.110 - Диаграмма классов

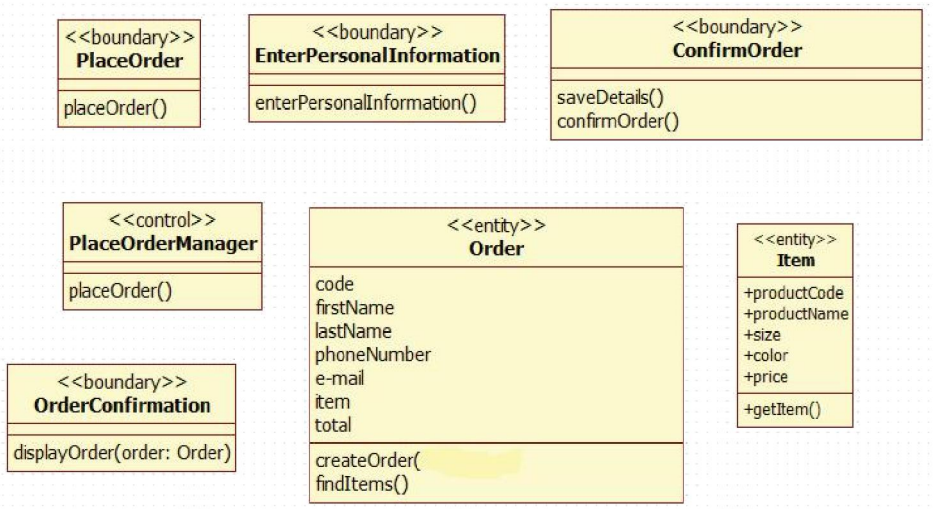


Рисунок 3.111 – Диаграмма классов с атрибутами и операциями

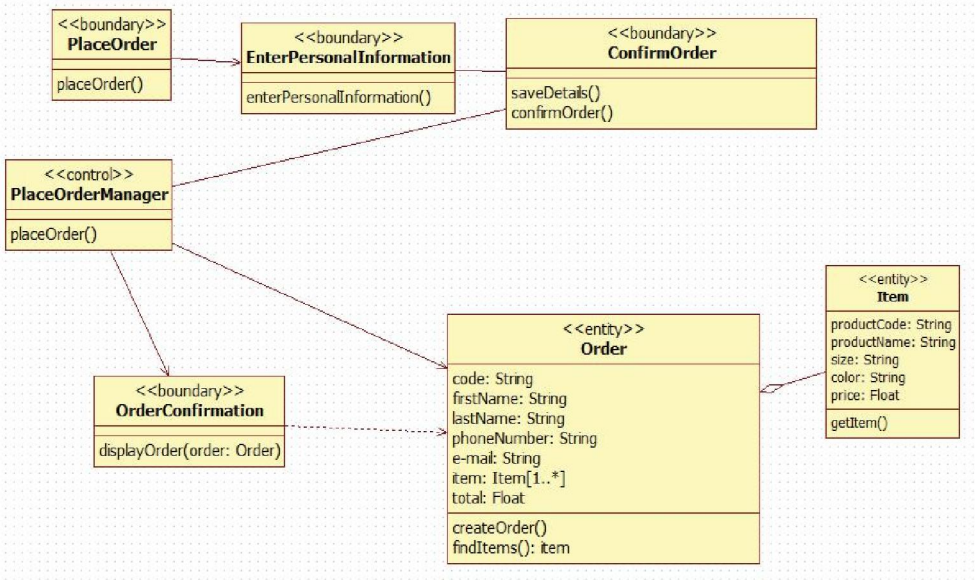


Рисунок 3.112 – Диаграмма классов с отношениями

Созданные ранее классы сценария Оформление заказа сгруппированы по пакетам: Граничные классы, Классы-сущности, Управляющие классы (рис. 3.113).

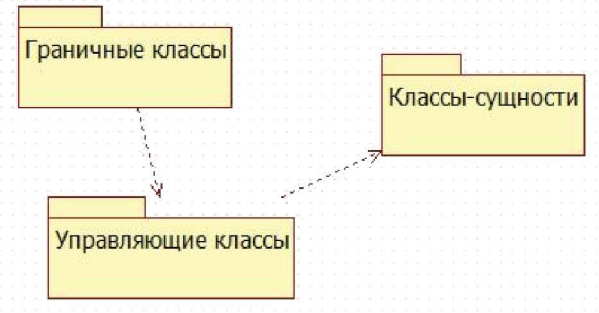


Рисунок 3.113 – Диаграмма пакетов

Оформление заказа включает указание своих личных контактных данных, электронной почты и оплату заказа. Оформление начинается из корзины покупателя, когда он выбирает опцию «Оформить заказ», рис. 3.114

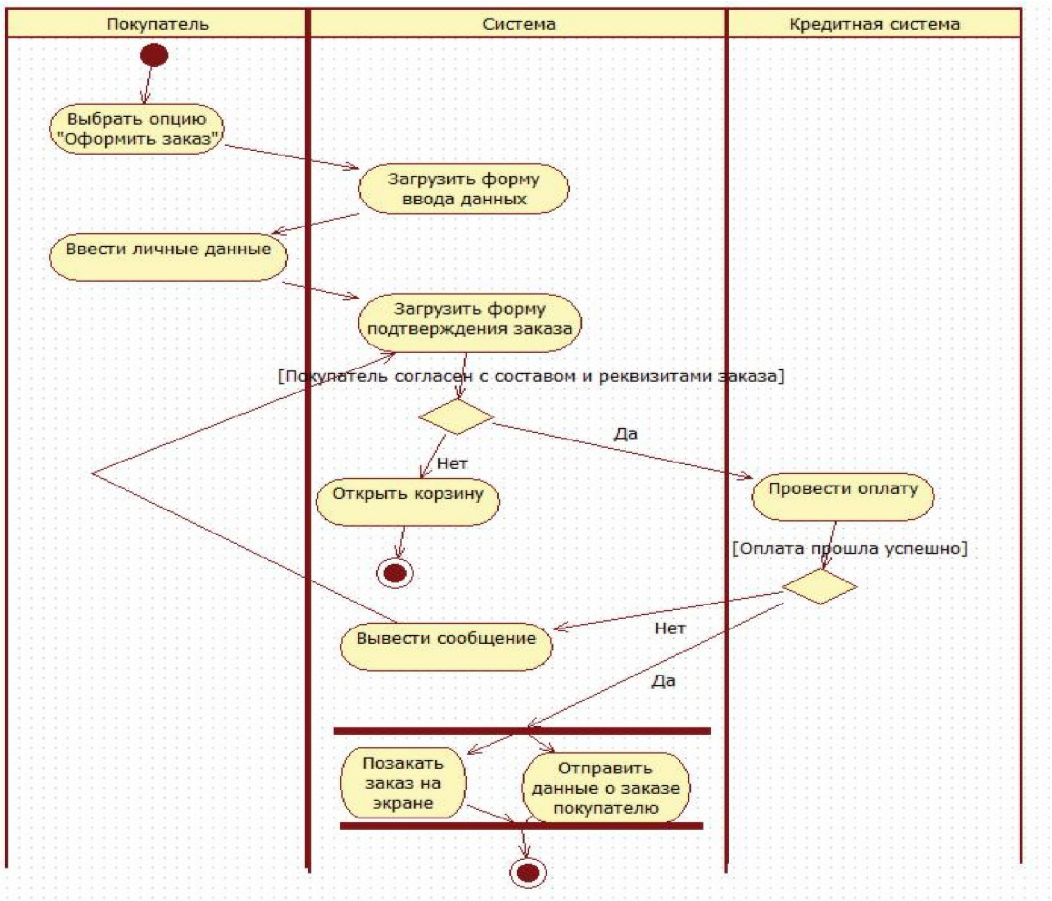


Рисунок 3.114 – Диаграмма деятельности

Покупатель выбирает опцию «Оформить заказ», затем открывается форма ввода личных данных покупателя и его кредитной карты (EnterPersonalInformation), на ней покупатель вводит свое имя, адрес, телефон, адрес электронной почты и кредитные данные. Информация принимается и открывается форма подтверждения заказа (ConfirmOrder). Фокус управления передается некоторому управляющему объекту (PlaceOrderManager), который обращается к внешней кредитной системе (Credit System) для проведения платежа. Если платеж прошел успешно, то PlaceOrderManager посылает сообщение объекту Заказ (Order), затем вызывает форму подтверждения заказа (OrderConfirmation). Объект Заказ (Order) обращается к объектам Товар (Item) для того, чтобы получить информацию о товарах и создает заказ.

Заказ оформляется покупателем, затем находится На обработке в отделе продаж, после передается на комплектацию кладовщику, если покупатель получил заказ, то это событие вызывает переход из состояния Укомплектован в состояние Выдан. Если [Покупатель не забрал заказ в течение 2 недель], то заказ будет Расформирован, покупателю возвращаются его деньги.

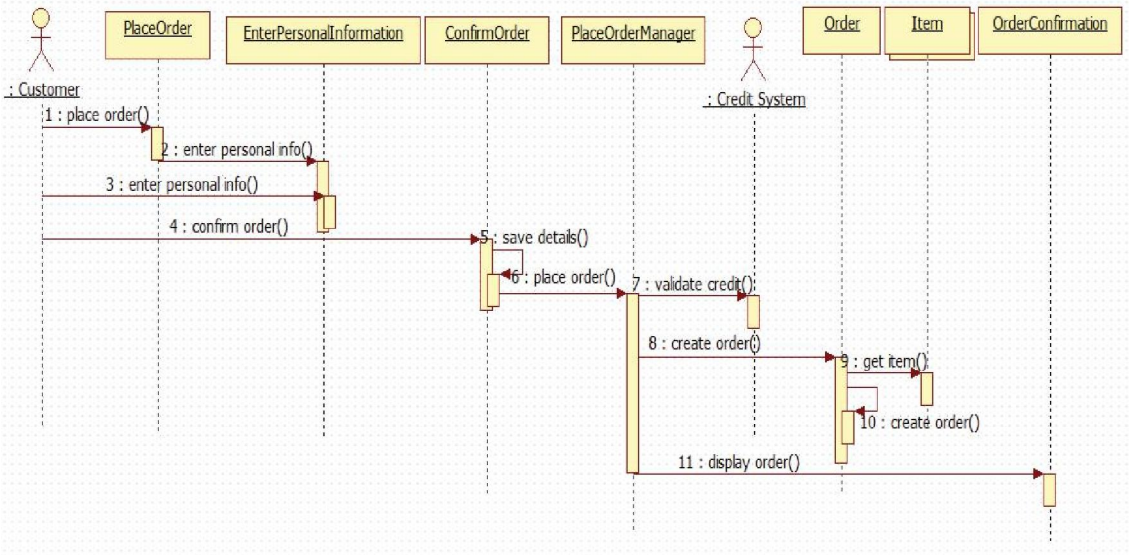


Рисунок 3.115 – Диаграмма последовательности

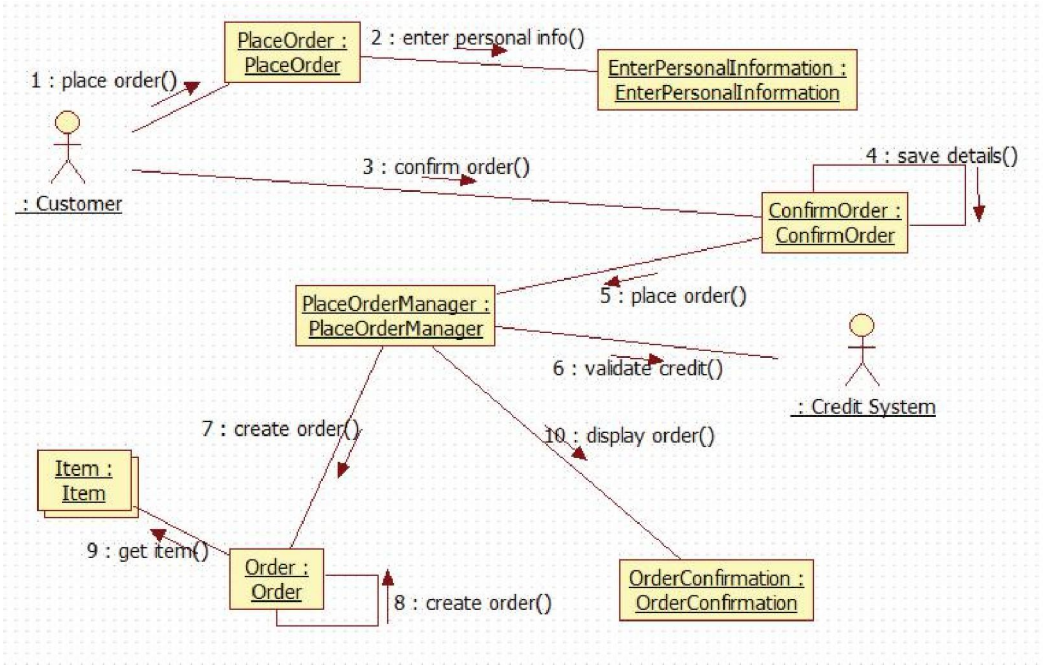


Рисунок 3.116 – Диаграмма кооперации

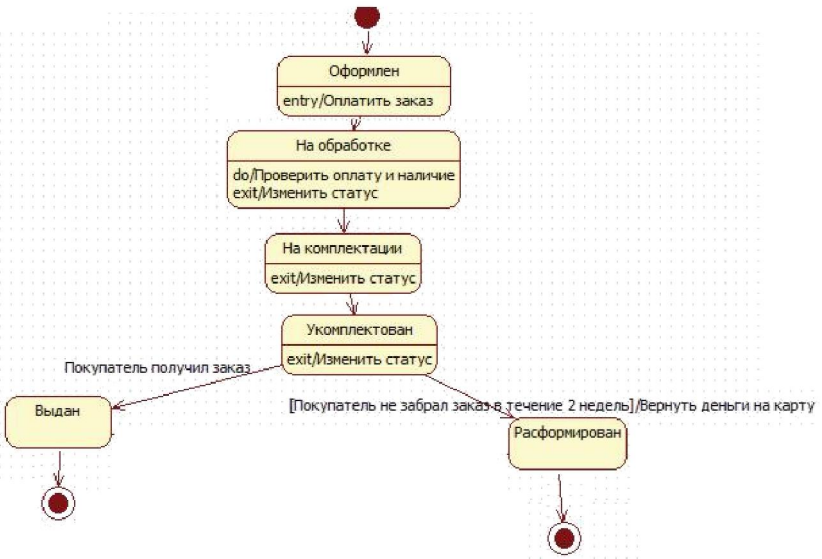


Рисунок 3.117 – Диаграмма состояний

Список литературы

1 ГОСТ 34.003-90 «Автоматизированные системы. Термины и определения».

2 ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы».

3 ГОСТ 34.601-90 «Автоматизированные системы. Стадии создания».

4 Вендров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. - 2-е изд., перераб. и доп. – М.: Финансы и статистика, 2006. – 544 с.

5 Силич, М. П. Создание модели деятельности организации с помощью инструментального средства "Design/IDEF": Методические указания для выполнения лабораторных работ по дисциплине «Теория организации» [Электронный ресурс] / Силич М. П. – Томск: ТУСУР, 2011. – 42 с.

6 Черемных СВ. Моделирование и анализ систем. IDEF-технологии: практикум / С.В.Черемных, И.О. Семенов, В.С. Ручкин. – М.: Финансы и статистика, 2006. – 192 с.

7 Калашян А.Н., Каляное Г.Н. Структурные модели бизнеса: DFD-технологии. – М.: Финансы и статистика, 2003.

8 Визуальное моделирование систем в StarUML: Учебное пособие/ А.В. Каюмова. Казань. – Казанский федеральный университет, 2013. – 104 с.

Приложение

Варианты индивидуальных:

1 Агентство по трудоустройству.
2 Обучающий центр.
3 Туристическая фирма.
4 Мастерская по ремонту бытовых приборов.
5 Рекламное агентство.
6 Редакция газеты.
7 Фирма по оказанию полиграфических услуг.
8 Гостиница.
9 Агентство по недвижимости.