МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)

Колледж экономики, управления и права

Методические указания
по организации
практических занятий и самостоятельной работы студентов
по МДК.01.02
Методы и средства проектирования информационных систем

Специальности

09.02.04 Информационные системы (по отраслям)

Ростов-на-Дону
2017

Методические указания по МДК.01.02 Методы и средства проектирования информационных систем разработаны с учетом ФГОС среднего профессионального образования специальности 09.02.04 Информационные системы (по отраслям), предназначены для студентов и преподавателей колледжа.

Методические указания определяют этапы выполнения работы на практическом занятии, содержат рекомендации по выполнению индивидуальных заданий и образцы решения задач, а также список рекомендуемой литературы.

Составитель (автор): Д.А. Морозюк преподаватель колледжа ЭУП

Рассмотрены на заседании предметной (цикловой) комиссии специальности 09.02.04 Информационные системы (по отраслям)

Протокол № 1 от 29 августа 2017 г

Председатель П(Ц)К специальности _____ С.В. Шинакова
личная подпись

и одобрены решением учебно-методического совета колледжа.

Протокол № 1 от 29 августа 2017 г

Председатель учебно-методического совета колледжа
_____ С.В. Шинакова
личная подпись

СОДЕРЖАНИЕ

Практическая работа №1

Проектирование информационной модели предметной области в нотации IDEF1X с помощью MS Visio

1. Цель работы

Целью работы является освоение технологии построения информационной модели логического и физического уровней в нотации IDEF1X с использованием пакета Microsoft Visio.

2. Задачи

Основными задачами практической работы являются:

- приобретение студентами навыков построения информационной модели логического уровня;
- нормализации полученной модели;
- построения информационной модели физического уровня.

3. Краткие теоретические сведения

3.1. Понятие информационной модели. Уровни информационной модели

Методология IDEF1X – язык для семантического моделирования данных, основанный на концепции «сущность-связь».

Различают два уровня информационной модели: **логический** и **физический**.

Логическая модель позволяет понять суть проектируемой системы, отражая логические взаимосвязи между сущностями.

Различают 3 подуровня логического уровня модели данных, отличающиеся по глубине представления информации о данных:

- диаграмма сущность-связь (Entity-Relationship Diagram (ERD);
- модель данных, основанная на ключах (Key Based Model (KB);
- полная атрибутивная модель (Fully Attributed Model (FA).

Физическая модель отражает физические свойства проектируемой базы данных (типы данных, размер полей, индексы). Параметры физической информационной модели зависят от выбранной системы управления базами данных (СУБД).

3.2. Основные элементы информационной модели логического уровня

3.2.1. Сущности и атрибуты

Сущность – это множество **реальных** или **абстрактных объектов** (людей, предметов, документов и т.п.), **обладающих общими атрибутами или характеристиками**. Любой объект системы может быть представлен только одной сущностью, которая должна быть уникально идентифицирована. *Именование сущности* осуществляется с помощью *существительного в единственном числе*. При этом имя сущности должно отражать **тип** или **класс** объекта, а не его **конкретный экземпляр** (например, **Студент**, а не **Петров**) (рис. 1).

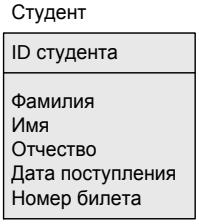


Рисунок 1 – Графическое представление сущности «Студент» в MS Visio

Любая сущность характеризуется набором атрибутов (**свойств**).

Атрибут сущности – характеристика сущности, то есть свойство реального объекта. Например, на рис. 1 атрибутами сущности **«Студент»** являются **«ID студента»**, **«Фамилия»**, **«Имя»**, **«Отчество»**, **«Дата поступления»** и **«Номер билета»**.

В свою очередь, *атрибуты сущности* делятся на 2 вида: *собственные* и *наследуемые. Собственные* атрибуты являются уникальными в рамках модели. *Наследуемые* атрибуты передаются от сущности-родителя при установке связи с другими сущностями.

Первичный ключ (Primary Key, PK). Каждая сущность должна обладать *атрибутом* или *комбинацией атрибутов*, чьи значения *однозначно определяют* каждый *экземпляр сущности*. Эти атрибуты образуют *первичный ключ* сущности.

Внешний ключ (Foreign Key, FK). Если между двумя сущностями *имеется специфическое отношение* связи или *категоризации*, то *атрибуты*, входящие в *первичный ключ родительской* или *общей сущности*, *наследуются* в качестве *атрибутов сущностью-потомком* или *категориальной сущностью* соответственно. Эти атрибуты и

называются внешними ключами. Наследуемый атрибут может использоваться в сущности в качестве части или целого первичного ключа, альтернативного ключа или не ключевого атрибута.

3.2.2. Отношения в IDEF1X-модели

При построении информационной модели различают следующие типы отношений между сущностями: *идентифицирующее, не идентифицирующее, не специфическое (многие-ко-многим) и отношения категоризации*.

Мощность отношения служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней.

3.3. Нормализация данных

Нормализация – это процесс проверки и реорганизации сущностей и атрибутов с целью удовлетворения требований к реляционной модели данных. Процесс нормализации сводится к последовательному приведению структур данных к нормальным формам – формализованным требованиям к организации данных.

Первая нормальная форма (1НФ). Сущность находится в первой нормальной форме тогда и только тогда, когда все атрибуты содержат атомарные значения. Среди атрибутов не должно встречаться повторяющихся групп, т.е. несколько значений для каждого экземпляра.

Вторая нормальная форма (2НФ). Сущность находится во второй нормальной форме, если она находится в первой нормальной форме, и каждый не ключевой атрибут полностью зависит от первичного ключа (не может быть зависимости от части ключа).

Третья нормальная форма (3 НФ). Сущность находится в третьей нормальной форме, если она находится во второй нормальной форме и никакой не ключевой атрибут не зависит от другого не ключевого атрибута (не должно быть зависимости между не ключевыми атрибутами).

4. Рекомендации по выполнению практического занятия

Практическая работа выполняется группой студентов (2-3 человека) в пакете Microsoft Visio.

Данное занятие может выполняться на основе результатов функционального моделирования предметной области.

Отчет по практическому занятию выполняется в формате MS Word, который содержит экранные формы моделей согласно заданию.

5. Методика выполнения практического занятия

Упражнение 1. Построение логической информационной модели уровня «сущность-связь»

5.1. Составление пула – списка потенциальных сущностей

Информационная модель может быть построена на основе функциональной модели или без нее. Использование функциональной модели в качестве основы для информационного моделирования позволяет создать структуру базы данных, полностью соответствующей функциям предприятия. Названия всех интерфейсных дуг функциональной модели (выполненной в нотации IDEF0) заносятся в *пул* – список потенциальных сущностей. Только в данном случае информационная модель будет адекватна выполняемым функциям. Функциональная модель для рассматриваемого примера представлена в *приложении А*.

Список потенциальных сущностей (при использовании программного продукта MS Office Visio для функционального моделирования) должен быть составлен вручную. В случае использования CASE-средства *AllFusion Process Modeler* отчет по интерфейсным дугам генерируется автоматически. Список потенциальных сущностей для рассматриваемого примера будет представлен таблицей вида (рис. 2).

Arrow Name
Варианты заданий
График
Графическая часть
Задание
Замечания, дополнения
Курсовая работа
Литература
Методические указания
Оценка за курсовую работу
Положение о курсовом проектировании
Пояснительная записка
Преподаватель
Расчеты
Список литературы
Студент

Рисунок 2 – Пул – список потенциальных сущностей

Теперь из этого списка необходимо выделить сущности, остальные интерфейсные дуги будут преобразованы в атрибуты сущностей.

В качестве сущностей выделим следующие:

1) задание;
2) пояснительная записка;
3) курсовая работа;
4) положение о курсовом проектировании;
5) студент;
6) преподаватель;
7) график;
8) методические указания.

5.2. Создание логической модели «сущность-связь»

1. Запустите MS Visio.
2. На закладке выбора шаблона выберите категорию *Программное обеспечение и базы данных* и в ней элемент *Схема модели базы данных*. Нажмите кнопку *Создать* в правой части экрана.
3. Установите необходимые параметры страницы (масштаб, ориентация страницы).
4. MS Visio поддерживает различные нотации моделей баз данных. Для того чтобы задать нотацию IDEF1X, необходимо выбрать пункты меню База данных → Параметры → Документ. В открывшемся окне на вкладке Общие установить переключатель в меню Набор символов на IDEF1X. Меню Имена, видимые на схеме, позволяет указать, какие имена атрибутов сущности будут отображены на диаграмме (концептуальные, физические или оба варианта одновременно). В данном случае для логического представления информационной модели необходимо выбрать пункт *Концептуальные имена* (рис. 3).

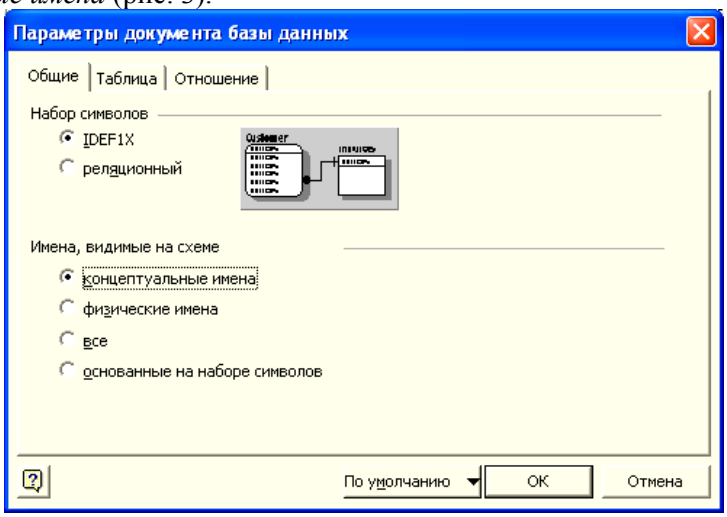


Рисунок 3 – Настройка параметров модели

В закладке *Отношение* окна *Параметры документа базы данных* в меню *Показывать* отметить галочкой пункт *Мощность*, в меню *Отображение вида* выбрать пункт *Показывать вербальную фразу*, снять галочку в пункте *Обратный текст* (рис. 4). Данные настройки позволят отобразить имя и мощность связи в модели.

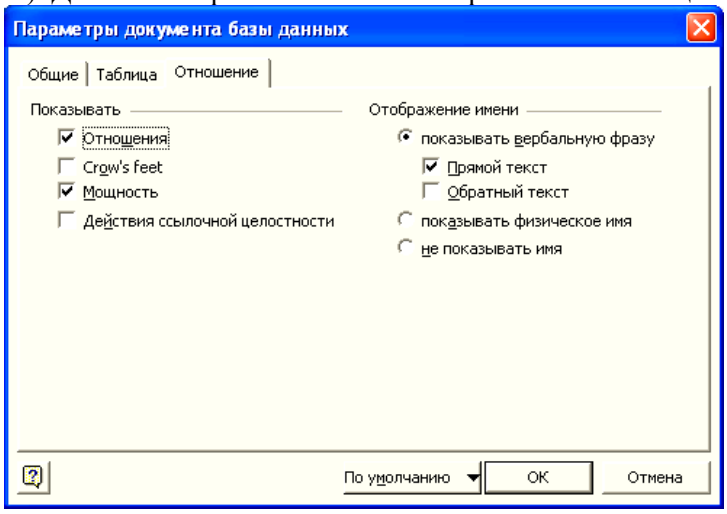


Рисунок 4– Настройка вида отношений информационной модели

5. Для того чтобы создать сущность, необходимо перетащить элемент  на рабочее поле. Переход в режим редактирования сущности осуществляется двойным щелчком по сущности или по нажатию правой кнопки мыши и выбора пункта меню *Свойства базы данных*.

Чтобы задать имя сущности, в окне *Свойства базы данных* нужно выбрать категорию *Определение*, снять галочку в пункте *Синхронизация имен при вводе* (в противном случае, физическое и логическое имя сущности будут совпадать, что по практическим соображениям не всегда удобно) и задать концептуальное имя сущности. Руководствуясь данным алгоритмом, создадим 8 сущностей, определенных в пункте 5.1 (см. рис. 5).

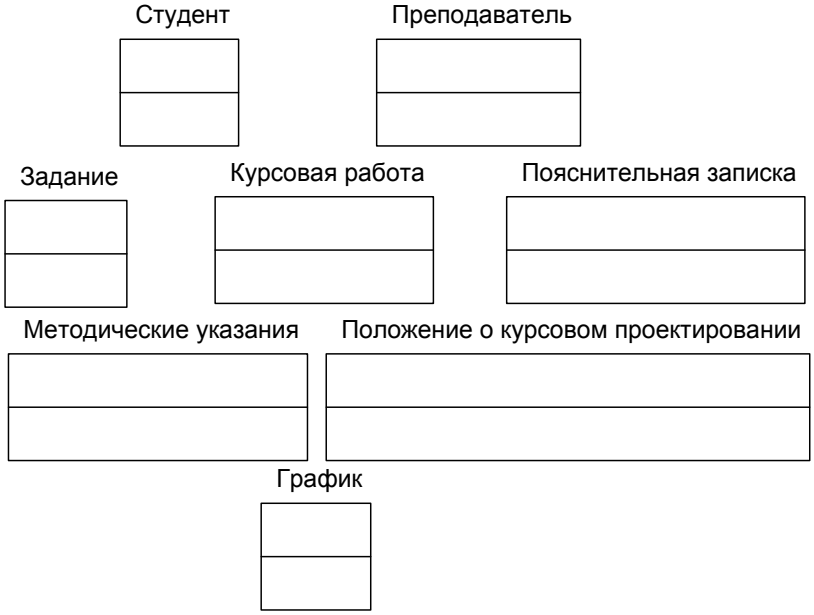


Рисунок 5– Сущности информационной модели логического уровня

6. Далее необходимо установить связи между сущностями.

Сначала составим описание предметной области на естественном языке.

Любой студент должен выполнить одну или несколько курсовых работ.

Каждая курсовая работа должна выполняться одним студентом (в идеале).

Каждая курсовая работа выполняется в соответствии с методическими указаниями и положением о курсовом проектировании.

Курсовая работа сдается по графику.

Курсовая работа оформляется в виде пояснительной записки.

Преподаватель проводит консультации, проверяет и ставит оценку за курсовую работу.

Таким образом, сформулируем имена связей:

СТУДЕНТ выполняет ***КУРСОВУЮ РАБОТУ***.

ПРЕПОДАВАТЕЛЬ проверяет ***КУРСОВУЮ РАБОТУ***.

КУРСОВАЯ РАБОТА выполняется в соответствии с ***ЗАДАНИЕМ***.

КУРСОВАЯ РАБОТА оформляется в виде ***ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ***.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ определяют требования к ***КУРСОВОЙ РАБОТЕ***.

КУРСОВАЯ РАБОТА организуется согласно ***ПОЛОЖЕНИЮ ПО КУРСОВОМУ ПРОЕКТИРОВАНИЮ***.

КУРСОВАЯ РАБОТА сдается по ***ГРАФИКУ***.

Во всех случаях сущность *Курсовая работа* является дочерней, за исключением связи с сущностью *Пояснительная записка*. Определим типы связей и построим модель (см. рис. 6). В дальнейшем можно будет подкорректировать связи между сущностями.

Чтобы установить **связи** между сущностями, необходимо перетащить на рабочую область элемент  , поднести один конец стрелки к родительской сущности, другой – к дочерней.

Примечание. При правильном связывании каждая сущность будет подсвечена красным цветом.

В MS Office Visio по умолчанию используется *не идентифицирующее* отношение. Чтобы изменить **тип связи**, необходимо двойным щелчком по связи открыть окно *Свойства базы данных* и в категории в категории *Прочее* указать тип отношения (идентифицирующее, не идентифицирующее). В этой же категории указывается мощность связи (см. рис. 6).

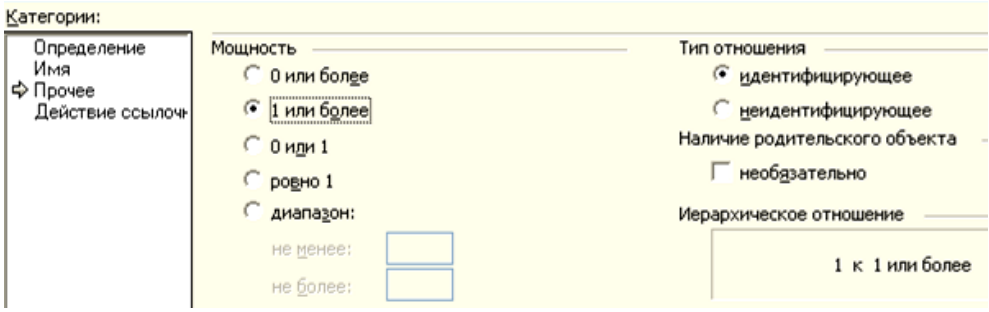


Рисунок 6 – Определение типа связи и мощности

Примечание. Кроме того, при не идентифицирующем отношении нужно указать, является ли наличие родительской сущности обязательным (т.е. может ли существовать экземпляр дочерней сущности, если не существует экземпляра родительской). Если наличие родительского объекта является необязательным, графически это отобразится в виде не закрашенного ромба со стороны родительской сущности.

Следующий шаг – в категории *Имя* в поле *Вербальная фраза* нужно указать имя отношения (рис. 7). Также можно указать имя связи в поле *Обратная фраза* для спецификации отношения потомок-родитель (в нашем случае обратная фраза отображаться не будет).

Примечание. Все изменения при закрытии окна свойств сохраняются автоматически.

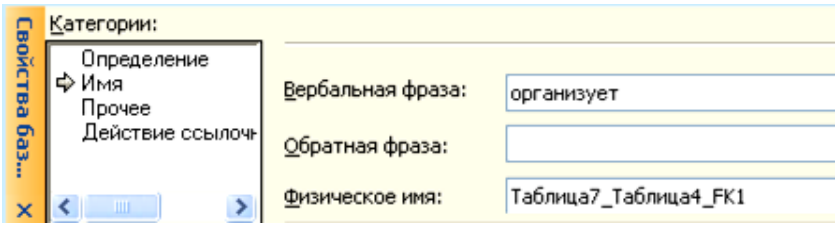


Рисунок 7 – Определение имени отношения

После определения имен, типов связей и задания мощностей получим информационную модель, представленную на рис. 8.

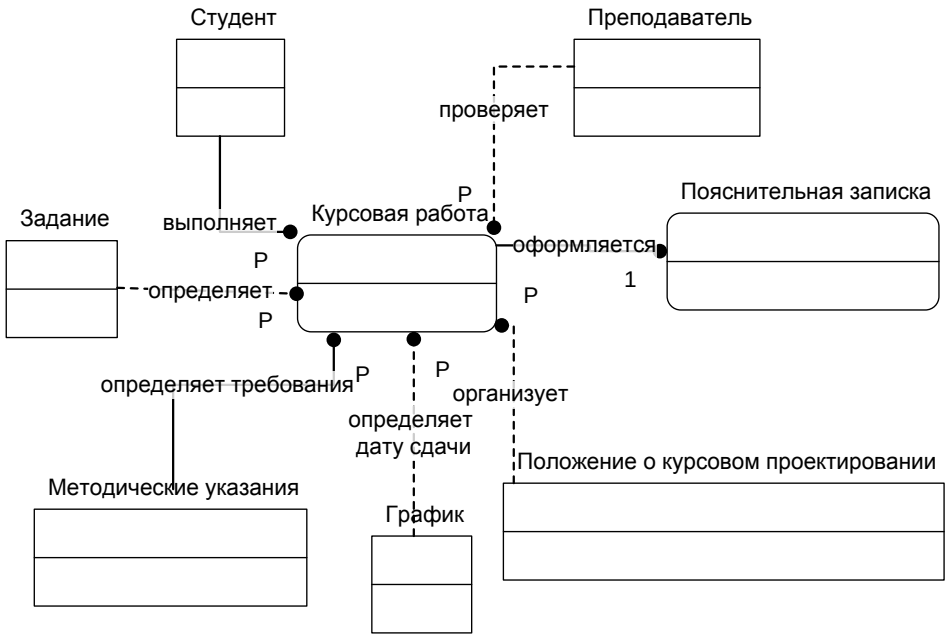


Рисунок 8 – Информационная модель уровня «сущность-связь»

Упражнение 2. Разработка логической модели данных, основанной на ключах

1. Необходимо определить ключевые атрибуты для каждой сущности, обращая внимание на то, что дочерние сущности наследуют ключевые атрибуты от родительских (см. рис. 10).

Для этого двойным щелчком мыши по сущности откроем окно редактирования ее свойств, перейдем в категорию *Столбцы*, по нажатию кнопки *Добавить* введем имя поля (например, для сущности *Задание* ключевым атрибутом будет являться *Вариант задания*). Чтобы сделать атрибут ключевым, необходимо отметить галочкой пункт *PK* (рис. 9). Данное поле становится обязательным автоматически.

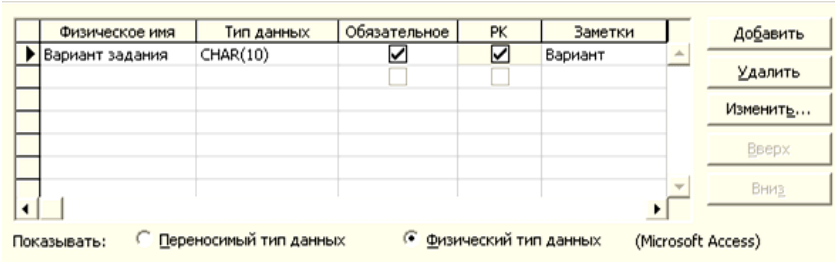


Рисунок 9 – Определение ключевого атрибута

Аналогичным образом зададим ключевые атрибуты для всех сущностей информационной модели. Результат представлен на рис. 10.

Как видно из рисунка 10 по сравнению с информационной моделью уровня «сущность-связь», был изменен тип связи между сущностями *Методические указания* и *Курсовая работа*, поскольку ключевые атрибуты сущности *Методические указания* для сущности *Курсовая работа* будут являться избыточными (зная номер зачетной книжки, можно узнать специальность и курс, на котором учится студент).

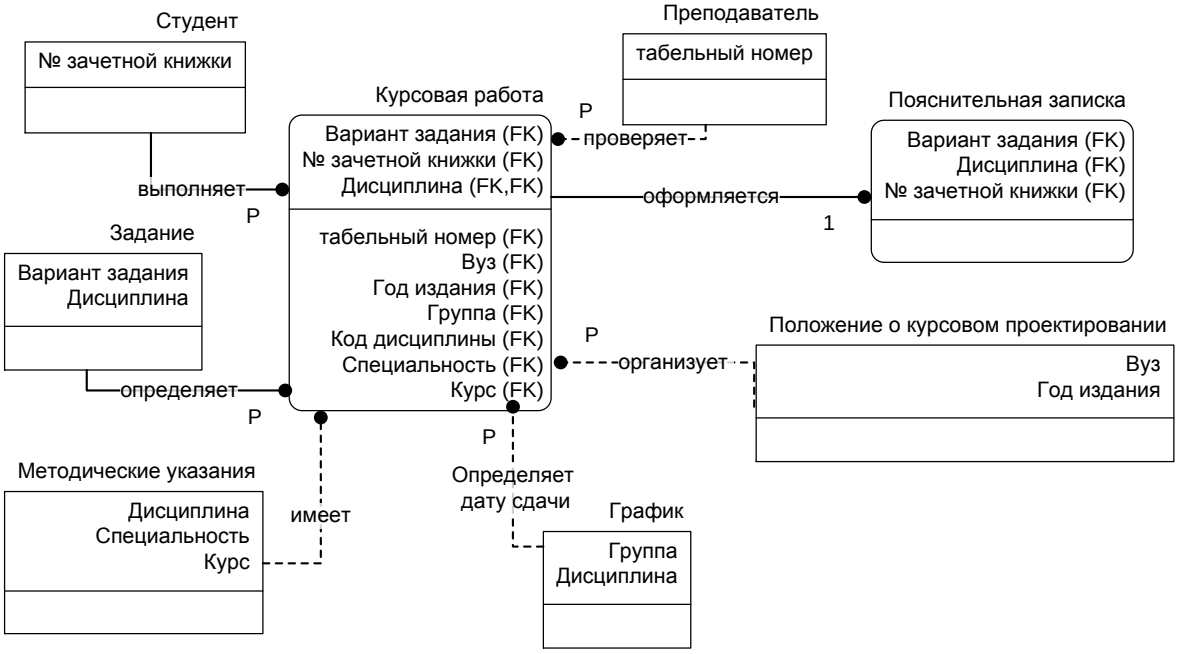


Рисунок 10 – Информационная модель с ключевыми атрибутами

2. Кроме того, отметим, что три сущности (Задание, График, Методические указания) содержат одинаковые атрибуты Дисциплина. Это является некорректным. Чтобы устранить данную ошибку, выделим одноименную сущность и свяжем ее идентифицирующими связями с вышеуказанными сущностями (рис. 11).

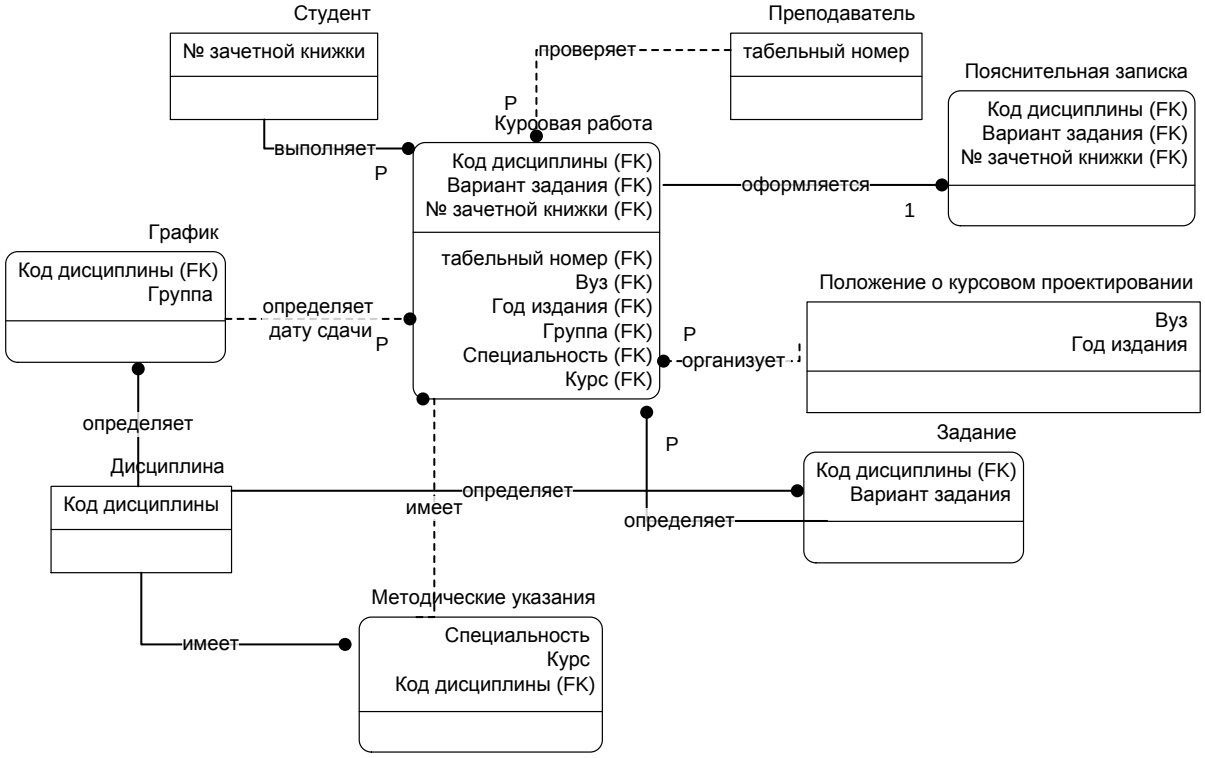


Рисунок 11 – Скорректированная информационная модель, основанная на ключах

Упражнение 3. Создание полной атрибутивной модели

Для того чтобы получить полную атрибутивную модель, необходимо дополнить сущности не ключевыми атрибутами. Дополненная модель представлена на рисунке 11.

Примечание. Если атрибут не является обязательным, нужно убедиться, что в окне *Свойства базы данных* в категории *Столбцы* в пункте *Обязательное* не стоит галочка. Не обязательные к заполнению атрибуты справа от имени имеют пометку *(О)*.

Упражнение 4. Нормализация полной атрибутивной модели

1. Проверим, все ли атрибуты имеют атомарные значения, т.е. среди атрибутов не должно встречаться повторяющихся групп, нескольких значений для каждого экземпляра (например, номер телефона_1, номер телефона_2). Видим, что атрибут *Авторы* в сущности *Методические указания* не удовлетворяет требованиям 1 НФ (у методических указаний может быть несколько авторов). Необходимо выделить сущность, которая будет содержать сведения об авторах методических указаний. Поскольку авторами всегда являются преподаватели вузов, новую сущность выделять не имеет смысла, свяжем сущности *Методические указания* и *Преподаватель*, предварительно удалив атрибут *Авторы*. Остальные атрибуты соответствуют 1 НФ. Атрибутивная модель, приведенная к 1 НФ, представлена на рис. 12.

Рисунок 12 – Полная атрибутивная модель

2. Приведем модель ко 2 НФ. Проверим, все ли атрибуты зависят от составного ключа, а не от его части. Проверка показала, что все не ключевые атрибуты сущностей полностью зависят от составного ключа. Значит, модель удовлетворяет требованиям 2 НФ.

3. Проверим, есть ли транзитивная зависимость между не ключевыми атрибутами. Проверка показала, что взаимозависимости между не ключевыми атрибутами нет. Таким образом, модель, представленная на рисунке 13, приведена к 3 НФ.

Примечание. К нормализации относились также действия, выполненные в п. 2 упражнения 2.

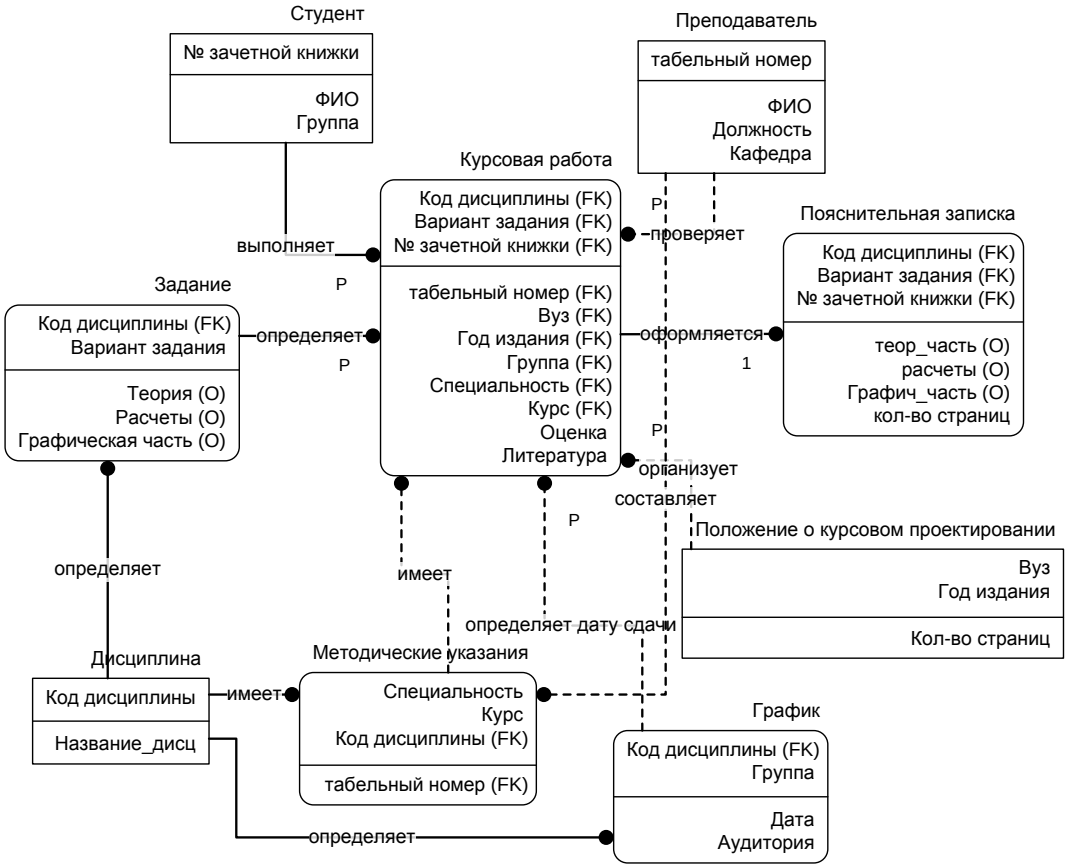


Рисунок 13 – Информационная модель, приведенная к 1 НФ

<u>Упражнение 5</u>. Создание физической модели

1. Необходимо переключиться на физический уровень представления информационной модели. Для этого нужно выбрать пункты меню *База данных → Параметры → Документ*. В открывшемся окне на вкладке *Общие* установить переключатель в меню *Имена, видимые на схеме*. В данном случае для физического представления информационной модели необходимо выбрать пункт *Физические имена* (рис. 14).

2. В закладке *Таблица* окна *Параметры документа базы данных* в меню *Отображать* выбрать пункт *Вертикальные линии,* в меню *Типы данных – Показывать физические* и в меню *Порядок – Физический порядок* (рис. 15).

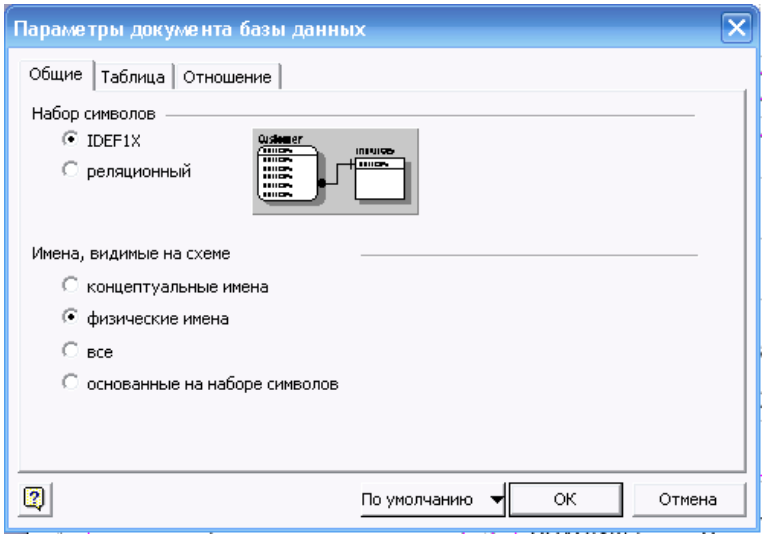


Рисунок 14 – Настройка параметров модели

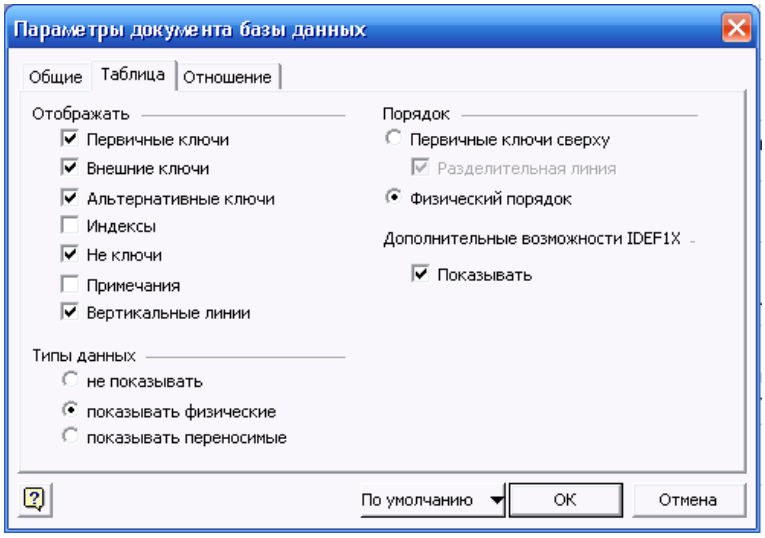


Рисунок 15 – Настройка параметров отображения сущности

3. В закладке Отношение окна Параметры документа базы данных в меню Отображение вида выбрать пункт Показывать физическое имя (рис. 16).

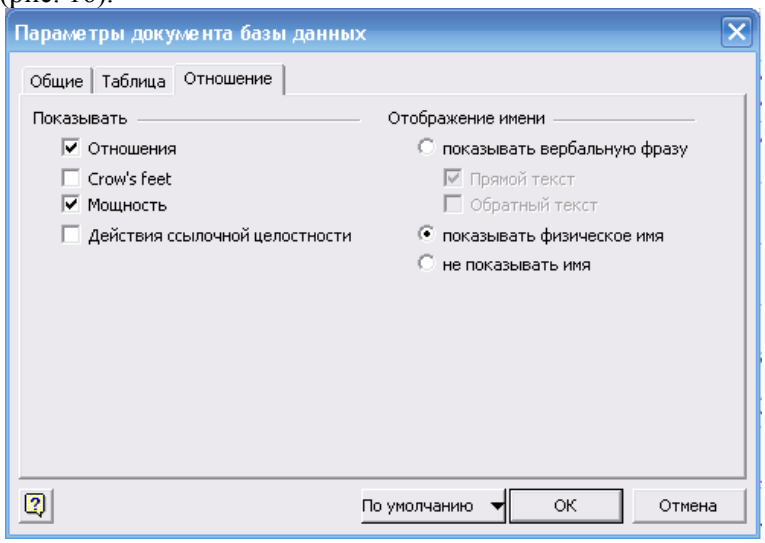


Рисунок 16 – Настройка вида отношений информационной модели

По окончании настройки документа информационная модель будет выглядеть, как представленная на рис. 17.

4. Для каждого атрибута (поля) необходимо определить тип данных.

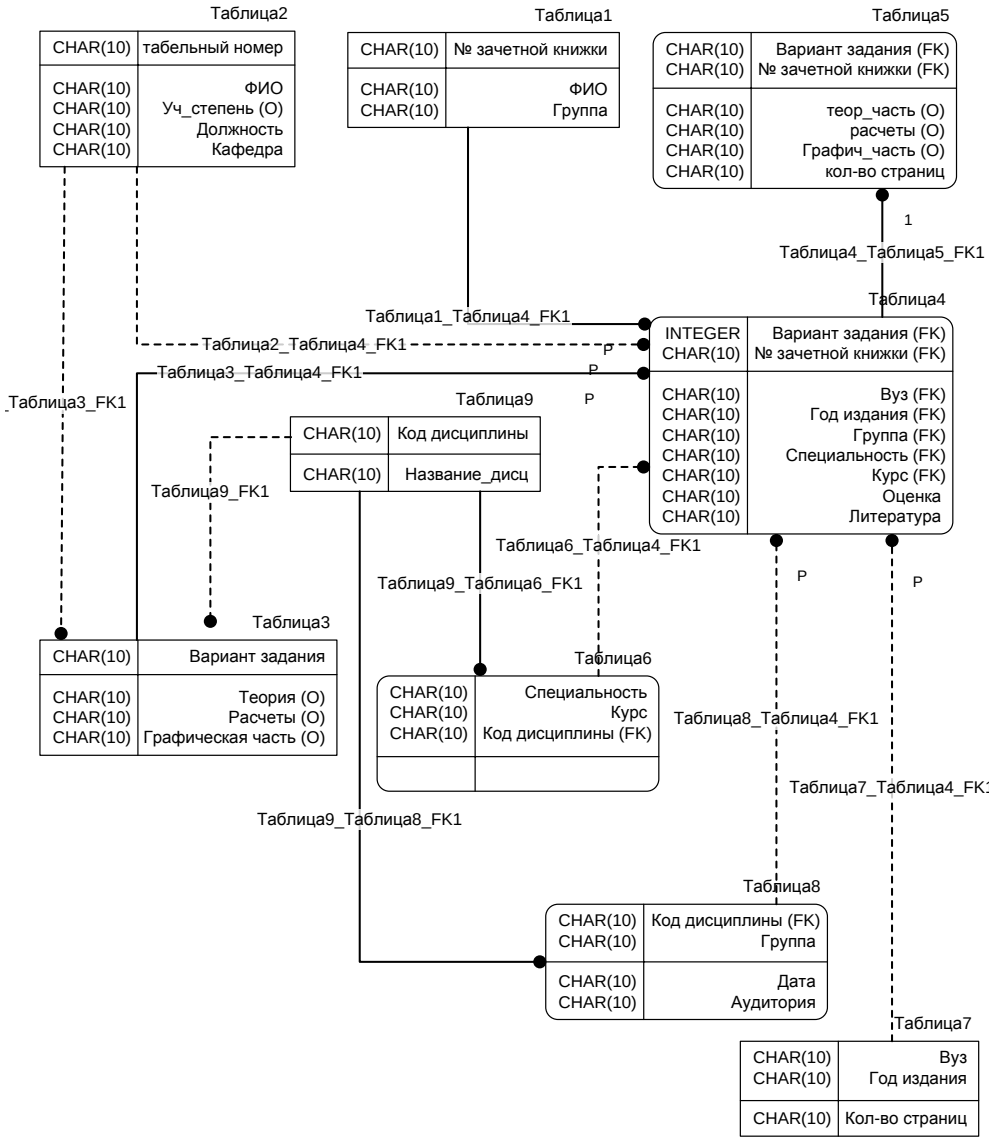


Рисунок 17 – Вид физической модели

Типы данных можно представить в виде правил, ограничивающих вид сведений, которые могут быть введены в каждый столбец таблицы базы данных. Например, чтобы в поле, которое предназначено только для дат, нельзя было ввести имя, этому полю назначается тип данных «Дата».

Примечание *(Выбор между переносимыми и физическими типами данных).*

Переносимые типы данных — это обобщенные типы данных, соответствующие в разных системах баз данных простым, совместимым между собой физическим типам.

Физические типы данных — это типы данных, поддерживаемые целевой базой данных.

Щелкните сущность, содержащую атрибуты, для которых требуется установить типы данных.

В окне *Свойства базы данных* в списке *Категории* выберите вариант *Столбцы*.

Под списком столбцов установите переключатель в положение *Физический тип данных*.

В группе *Тип данных* для каждого атрибута выберите необходимый вариант из множества альтернатив (рис. 18). Описание типов данных приведено в *Приложении Б*.

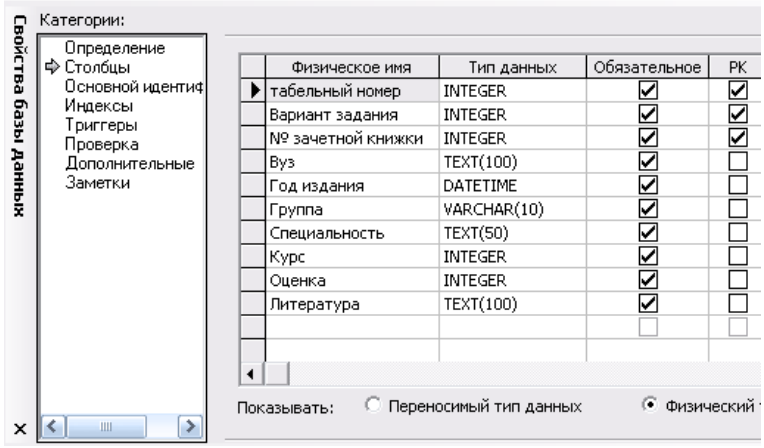


Рисунок 18– Определение типа данных атрибутов сущности

После того, как будут выполнены все действия, физическая модель будет выглядеть, как показано на рис. 19.

Таким образом, проделав все вышеперечисленные действия, получим информационную модель физического уровня, на основе которой может быть сгенерирована схема БД (в нашем случае в MS Office Access).

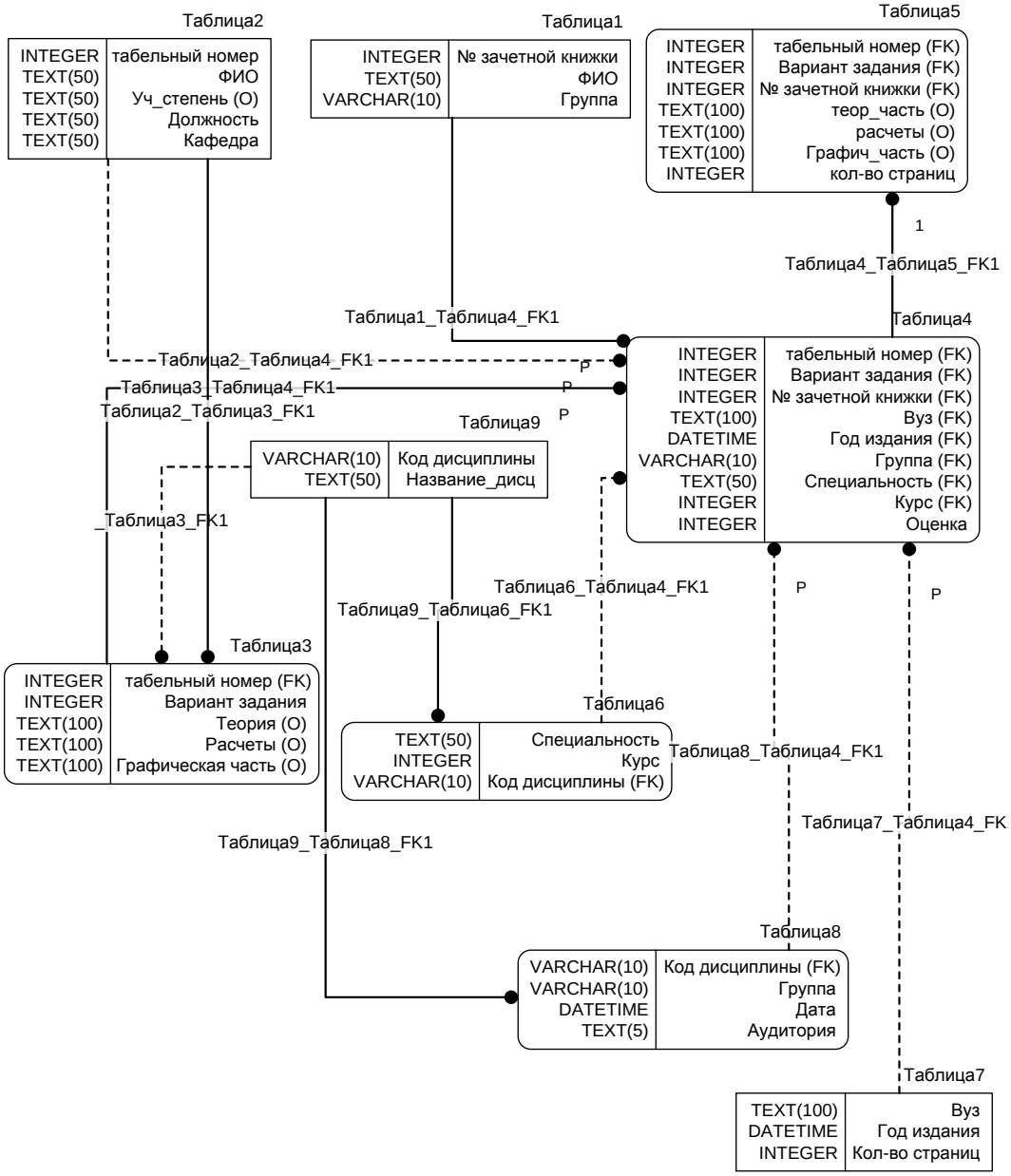


Рисунок 19 – Физическая модель базы данных

6. Задание

В соответствии с вариантом задания, определенным преподавателем, последовательно выполнить следующие действия:

1) создать информационную модель логического уровня (выполнить упражнения 1-3). Минимальное количество сущностей – 4;

2) провести нормализацию полученной модели (упражнение 4);

3) на основе нормализованной логической модели построить информационную модель физического уровня (упражнение 5).

7. Варианты

1. Проектирование ИС «Отдел кадров»;
2. Проектирование ИС «Агентство аренды»;
3. Проектирование ИС «Аптека»;
4. Проектирование ИС «Ателье»;
5. Проектирование ИС «Аэропорт»;
6. Проектирование ИС «Библиотека»;
7. Проектирование ИС «Кинотеатр»;
8. Проектирование ИС «Поликлиника»;
9. Проектирование ИС «Автосалон»;
10. Проектирование ИС «Таксопарк».

8. Порядок выполнения работы

Для выполнения работы необходимо:

а) повторить правила техники безопасности при работе с вычислительной техникой;

б) изучить соответствующий раздел лекционного курса, а также теоретическую часть настоящего методического указания;

в) выполнить лабораторную работу согласно описанной в пункте 5 методике в соответствии с вариантом задания;

г) в соответствии с требованиями, приведенными в разделе 8 практикума, оформить отчет по лабораторной работе;

е) защитить лабораторную работу в соответствии с требованиями преподавателя.

9. Требования к содержанию и оформлению отчета

Отчет должен содержать:

1) титульный лист;
2) название практического занятия, цель;
3) пул – список потенциальных сущностей;
4) нормализованную информационную модель логического уровня;
5) информационную модель физического уровня;
6) выводы по проделанной работе.

Отчет должен быть представлен в бумажном виде.

10. Контрольные вопросы

1. Какие диаграммы позволяет строить нотация IDEF1X?
2. Для чего предназначена диаграмма «сущность-связь»?
3. Чем отличается полная атрибутивная модель от диаграммы «сущность-связь»?
4. Какие виды отношений существуют и чем они отличаются?
5. Что представляет собой нормализация?
6. В чем разница между логическим уровнем модели данных и физическим?

11. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вендров А. М. Практикум по проектированию программного обеспечения экономических информационных систем. : - М.: Финансы и статистика, 2004.-190 с.

2. Вендров А. М. Проектирование программного обеспечения экономических информационных систем -М.: Финансы и статистика, 2006.-543 с.

3. Маклаков С.В. Создание информационных систем с AllFusion Modeling Suite. – М.: ДИАЛОГ-МИФИ, 2005 – 432с.

ПРИЛОЖЕНИЕ А

Функциональная модель процесса «Выполнить курсовую работу»

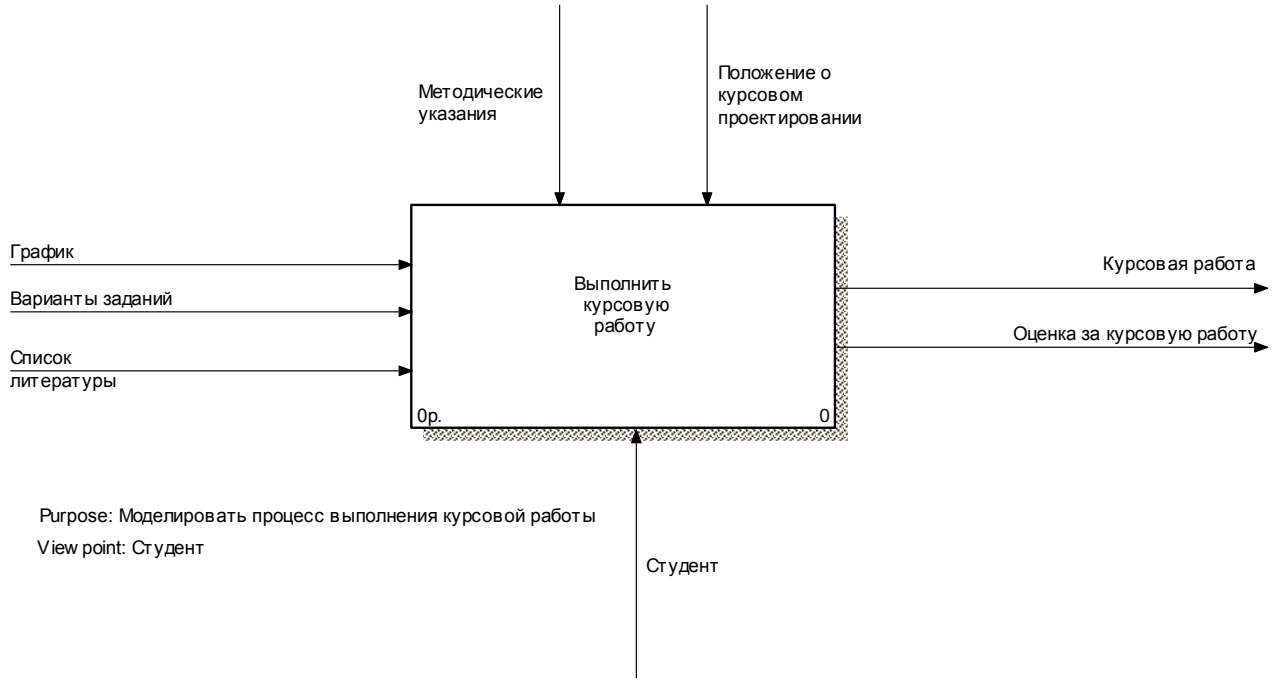


Рисунок А.1 – Контекстная диаграмма процесса выполнения курсовой работы

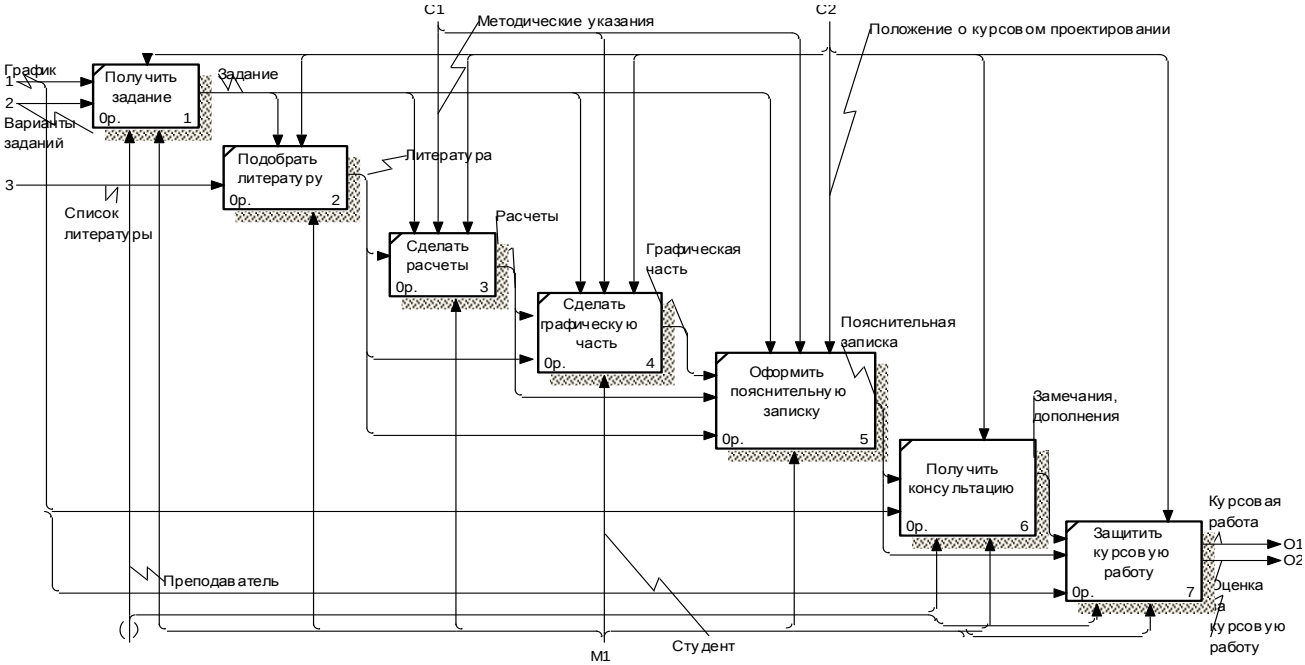


Рисунок А.2 – Диаграмма декомпозиция блока А0

ПРИЛОЖЕНИЕ Б

Типы данных Microsoft Office Access

1. Символьные типы

Символьные типы используются для представления как строк символов, так и отдельных символов.

Таблица Б.1 – Перечень символьных типов

Тип данных	Назначение	Размер
CHAR	Строковый тип	до 32767 байт. по умолчанию 1 байт
VARCHAR	Тоже, что и CHAR	
LONG VARCHAR	Символьный тип произвольной длины. Аналог MEMO-полям в dBase, FoxPro, Access	Длина произвольная. Ограничена максимальным размером файлов базы данных (2 гигобайта)
TEXT	Тоже, что и LONG VARCHAR	

2. Числовые типы

Числовые типы предназначены для обозначения целых, вещественных и денежных типов.

Таблица Б.2 – Перечень числовых типов

Тип данных	Диапазон значений	Точность - число знаков после запятой	Размер
INTEGER	от -2 147 483 648 до +2 147 483 647	0	4 байта
SMALLINT	от -32 768 до +32 767	0	2 байта
REAL	от -3.4 e-38 до 3.4 e+38	до 6	4 байта
DOUBLE	от -1.797 e-308 до +1.797 e+308	до 15	8 байт
DECIMAL	числа, состоящие из N цифр с M цифрами в дробной части. По умолчанию N=30, M=6	M	сколько требуется
NUMERIC	Тоже, что и DECIMAL		

3. Типы дата/время

Типы дата/время предназначены для хранения времени, дат и дат совместно с временем.

Таблица Б.3 – Форматы представления данных типа дата/время, определяемые по умолчанию

Тип данных	Формат, используемый по умолчанию
DATETIME	'YYYY-MM-DD HH:NN:ss.SSS'

- **YYYY** – четыре цифры, обозначающие год:
- **MM** – две цифры, обозначающие месяц:
- **DD** – две цифры, обозначающие день:
- **HH** – две цифры, обозначающие часы:
- **NN** – две цифры, обозначающие минуты:
- **ss** – две цифры, обозначающие секунды:
- **SSS** – три цифры, обозначающие доли секунд.

По умолчанию составляющие времени HH, NN, ss, SSS принимаются равными нулю, а DD - единице.

4. Двоичные типы

Двоичные типы предназначены для представления двоичных данных, включая изображения и другую информацию, не обрабатываемую собственными средствами СУБД.

Таблица Б4 -

Таблица Б.4 – Двоичные типы

Тип данных	**Назначение**	**Размер**
BIT	Тип для представления значений 0 и 1. Аналог полей типа Logical в dBase, FoxPro	1 байт
BINARY	Тоже, что и CHAR, за исключением операций сравнения. В отличии от CHAR, данные этогопо умолчанию 1 байт типа сравниваются на полное совпадение двоичных кодов байтов	до 32767 байт
LONG	Тип для представления двоичных данных произвольной длины	Длина произвольная. Ограничена максимальным размером файлов базы данных (2 гиго- байта)

Практическая работа №2

Функциональное моделирование предметной области в нотации IDEF0 с помощью MS Visio

1. Цель работы

Целью работы является получение навыков создания и редактирования функциональных моделей в нотации IDEF0 с помощью MS Visio.

2. Задачи

Основными задачами практической работы являются:
- приобретение студентами навыков построения функциональной модели;
- приобретение студентами навыков редактирования функциональной модели.

3. Краткие теоретические сведения

3.1. Основные сведения по методологии IDEF0

Модель в нотации IDEF0 представляет собой совокупность иерархически упорядоченных и взаимосвязанных диаграмм. Каждая диаграмма является единицей описания системы и располагается на отдельном листе.

Цель моделирования. Модель не может быть построена без четко сформулированной цели. Пример цели: «Описать функциональность предприятия с целью написания спецификаций ИС».

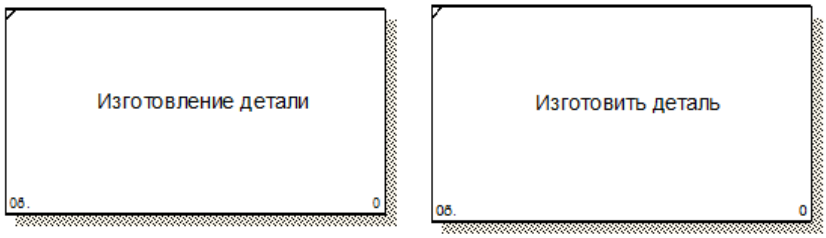
Точка зрения. Точку зрения можно представить как взгляд человека, который видит систему в нужном для моделирования аспекте. Как правило, выбирается точка зрения человека, ответственного за моделируемую работу в целом. Цель и точка зрения документируются.

Основные элементы IDEF0-модели

В основе методологии IDEF0 лежат 4 основных понятия:
- функциональный блок;
- интерфейсная дуга (стрелка);
- декомпозиция;
- глоссарий.

1. Функциональный блок

Функциональные блоки обозначают поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. Графически функциональные блоки изображаются в виде прямоугольников. Все блоки должны быть названы и определены. Имя функционального блока должно быть выражено сочетанием отглагольного существительного, обозначающего процесс, или глаголом (рис. 1):



а) б)

Рисунок 20 – Примеры работ

Определение функционального блока заносится в глоссарий или словарь работ (Activity Dictionary).

Все функциональные блоки модели нумеруются. Номер состоит из префикса и числа. Может использоваться префикс любой длины, но обычно используется префикс А. Контекстная (корневая) работа (функциональный блок) имеет номер А0.

2. Интерфейсная дуга (стрелка – Arrow)

Взаимодействие функциональных блоков с внешним миром и между собой описывается в виде интерфейсных дуг (стрелок). Стрелки представляют собой некую информацию и обозначаются существительными (например, «Заготовка», «Изделие») или именуемыми сочетаниями (например, «Готовое изделие»). Все стрелки должны быть определены. Определения заносятся в словарь стрелок – глоссарий (Arrow Dictionary).

В IDEF0 различают 4 типа стрелок (рис.2).

Каждая стрелка имеет свое расположение относительно функционального блока.

Рисунок 21 – Типы стрелок

Вход (Input) – материал или информация, которые используются или преобразуются работой для получения результата (выхода). Стрелка Input рисуется входящей в левую грань работы.

Управление (Control) – правила, стратегии, процедуры или стандарты, которыми руководствуется работа. Каждая работа должна иметь хотя бы одну стрелку управления. Рисуется как входящая в верхнюю грань работы.

Выход (Output) – материал или информация, которые производятся работой. Каждая работа должна иметь хотя бы одну стрелку выхода. Работа без результата не имеет смысла и не должна моделироваться. Изображается исходящей из правой грани работы.

Механизм (Mechanism) – ресурсы, которые выполняют работу, например, персонал предприятия, станки, устройства и т.д. Рисуется как входящая в нижнюю грань работы.

3. Глоссарий

Набор определений, ключевых слов и т.д., которые характеризуют каждый объект модели.

4. Декомпозиция

Разбиение системы на крупные фрагменты – функции, функции – на подфункции и т.д. до конкретных процедур.

Модель может содержать 4 типа диаграмм:

- контекстную (в каждой модели может быть только 1 контекстная диаграмма);
- декомпозиции;
- дерева узлов;
- только для экспозиции (FEO).

Контекстная диаграмма является вершиной древовидной структуры диаграмм и представляет собой общее описание системы и ее взаимодействия с внешней средой.

После описания системы в целом проводится разбиение ее на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов – диаграммами декомпозиции. После декомпозиции контекстной диаграммы проводится декомпозиция каждого большого фрагмента системы на более мелкие и т.д., до достижения нужного уровня подробности описания.

Диаграмма дерева узлов показывает иерархическую зависимость работ, но не взаимосвязи между работами.

Диаграммы для экспозиции (FEO) строятся для иллюстрации отдельных фрагментов модели, для иллюстрации альтернативной точки зрения либо для специальных целей.

Все диаграммы имеют нумерацию. Контекстная диаграмма имеет номер А-0, декомпозиция контекстной диаграммы – номер А), остальные диаграммы-декомпозиции – номера по соответствующему узлу (например, А1, А2, А21 и т.д.).

3.2. Особенности MS VISIO

3.2.1 Для построения функциональной модели бизнес-процесса, используя MS Visio, необходимо запустить программу.

В открывшейся программе выбрать: Файл – Фигуры – Блок-схема – Фигуры схемы IDEF0.

3.2.2 Используемые блоки для построения функциональной модели:

Блок заголовка – рамка, которую необходимо установить на весь лист и оформить в соответствии с правилами оформления диаграмм в нотации IDEF0

Блок текста необходим для описания точки зрения и цели на контекстной диаграмме.

Блок действия – для описания работ, рассматриваемых в процессе.

Одностороннее соединение – элемент изображения интерфейсных дуг, таких как вход/выход, механизм/управление.

Соединительная линия IDEF0 – объект для изображения интерфейсных дуг между работами в модели.

4. Методика выполнения

В качестве примера рассматривается процесс выполнения студентом курсовой работы (курсового проекта).

4.1. Создание контекстной диаграммы

1. Запустите MS Visio.
2. На закладке выбора шаблона выберите категорию *Блок-схема* и в ней элемент *Схема IDEF0*. Нажмите кнопку *Создать* в правой части экрана.
3. Окно программы примет вид, подобный рис. 3.

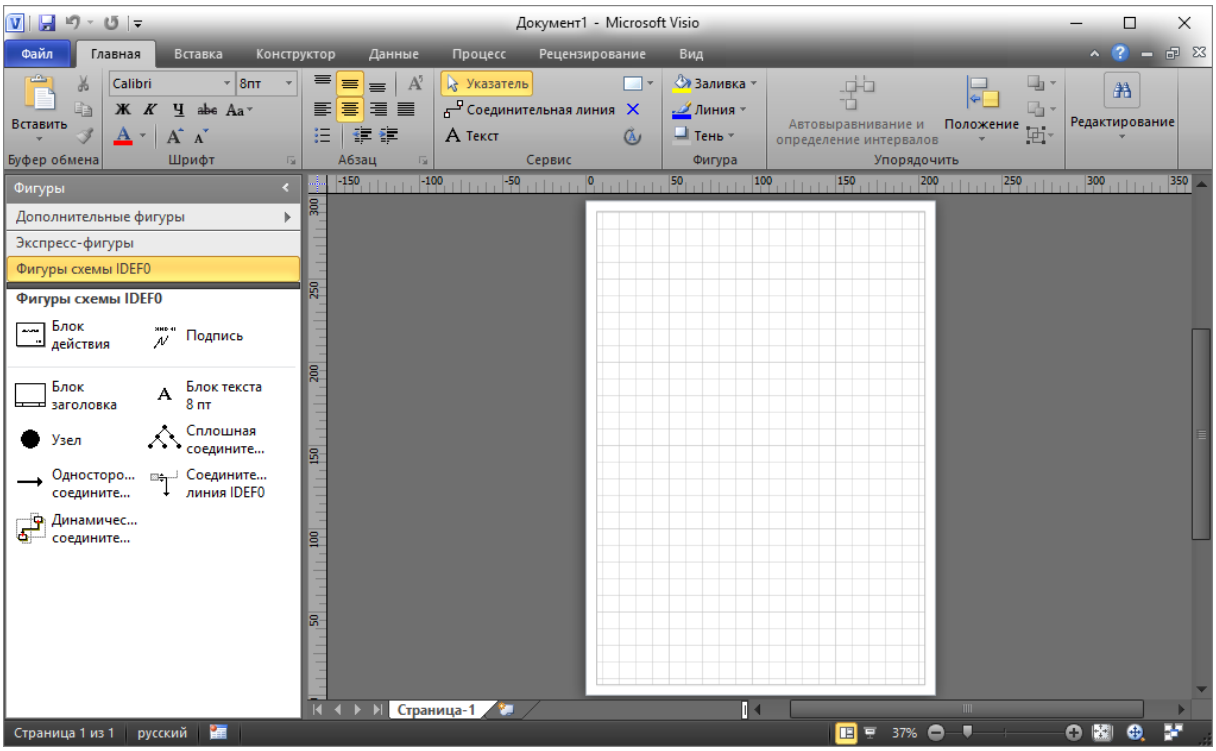


Рисунок 22 – Окно программы с выбранным шаблоном

4. Создание мастерской страницы.

4.1. Для удобства переведите страницу в альбомный вид: Конструктор – Параметры страницы – Ориентация – Альбомная;

4.2. Перетащите Блок заголовка на пустую страницу, удерживая нажатой правую кнопку мыши;

4.3. Заполнить поле «Заголовок» (рисунок 4), предложенное в открывшемся окне: внести номер контекстной диаграммы и имя рассматриваемого процесса, в данном случае: А-0 Выполнить курсовую работу;

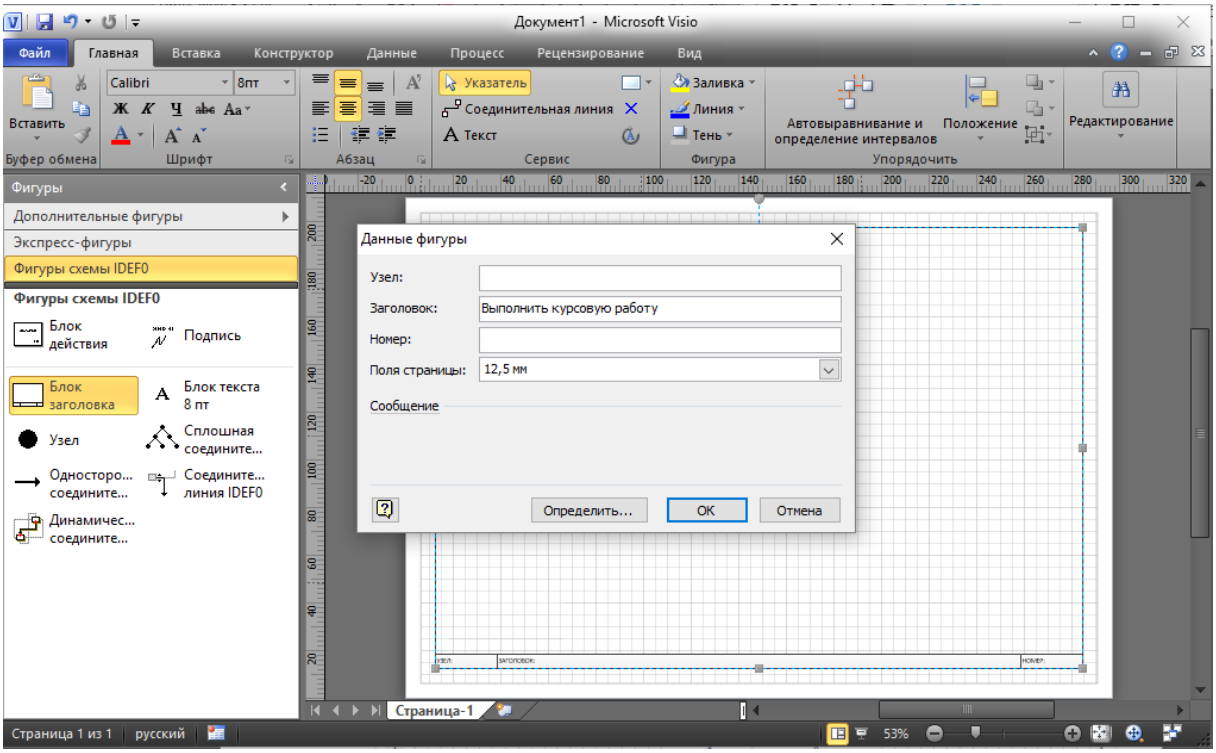


Рисунок 23 – Мастерская страница

4.4. Далее, имя заголовка фигуры «*Блок заголовка*» должно соответствовать номеру и названию задачи, декомпозиция которой будет изображена в данной области. Например: А1 Получить задание.

5. Определение цели и точки зрения.

5.1. С помощью кнопки *Блока текста* внесите текст в поле диаграммы – точку зрения и цель (рис. 5).

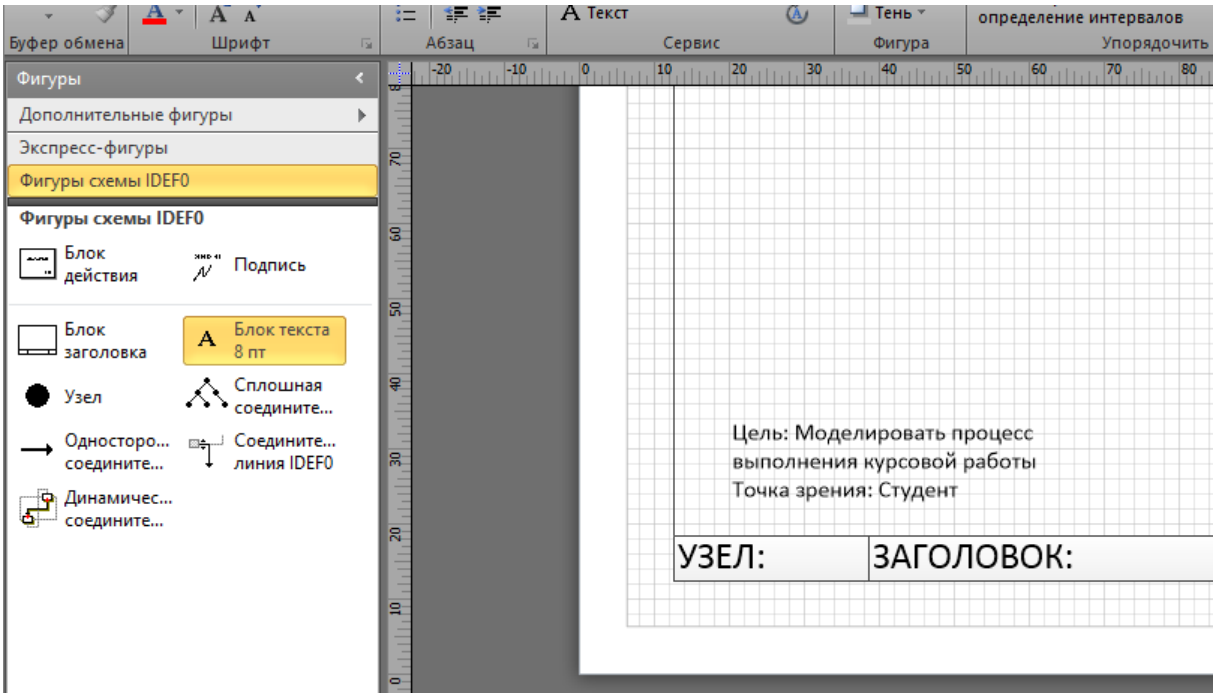


Рисунок 24 – Цель и точка зрения

6. В область диаграммы (поле *Блока заголовка*) внесите *Блок действия*. В открывшемся окне «Данные фигуры» внесите имя процесса и идентификатор процесса (рис. 6).

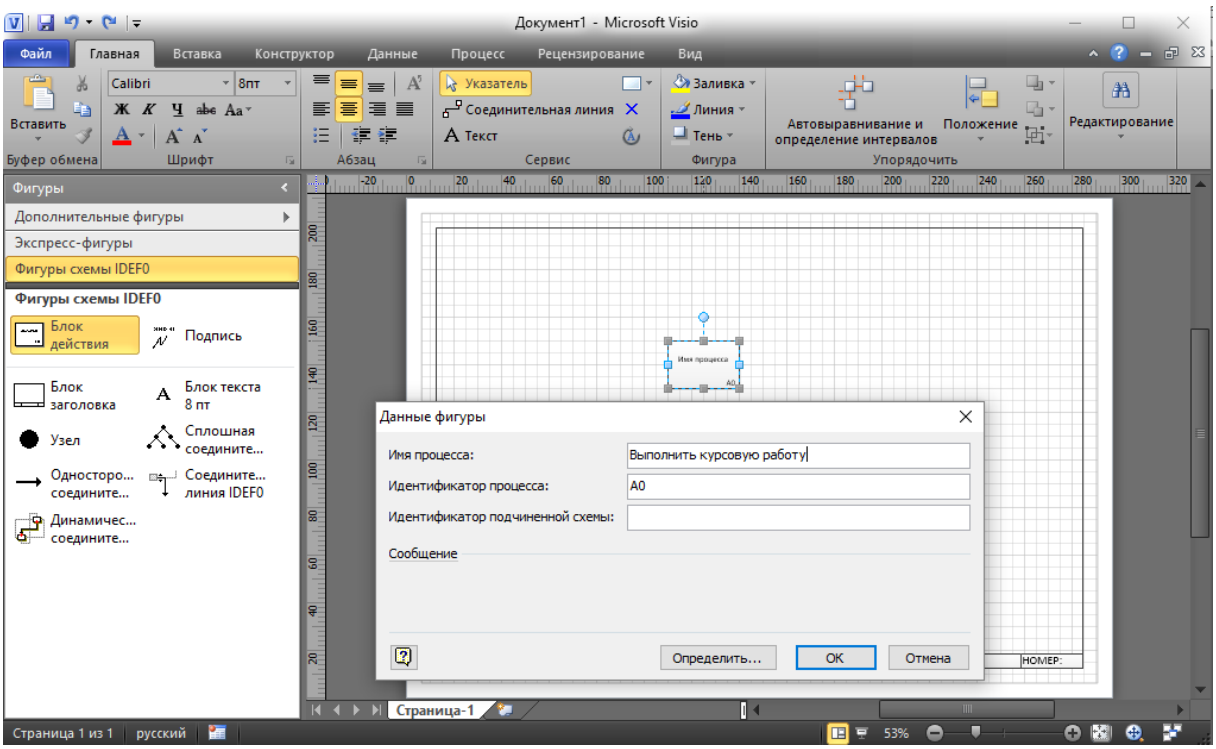


Рисунок 25 – Блок действия

7. С использованием блока *Односторонняя соединительная линия* создайте стрелки на контекстной диаграмме (табл. 1). Чтобы добавить текст необходимо дважды щелкнуть по стрелке.

Таблица 5 – Стрелки контекстной диаграммы

Имя стрелки (Arrow Name)	**Определение стрелки (Arrow Definition)**	**Тип стрелки (Arrow Type)**
График	График консультаций и сроки сдачи	Input
Список литературы	Источники информации для выполнения курсовой работы	Input
Варианты заданий	Список заданий на курсовую работу, подлежащий распределению между студентами	Input
Методические указания	Документ, содержащий указания по выполнению курсовой работы, описывающий содержание ее частей и основные требования	Control
Положение о курсовом проектировании	Документ, отражающий организационные требования по выполнению и сдаче курсовой работы	Control
Курсовая работа	Документ, являющийся основанием для получения оценки	Output
Оценка за курсовую работу	Результат выполнения курсовой работы	Output
Студент	Тот, кто выполняет курсовую работу	Mechanism

8. Результат выполнения предыдущих пунктов представлен на рис. 7.

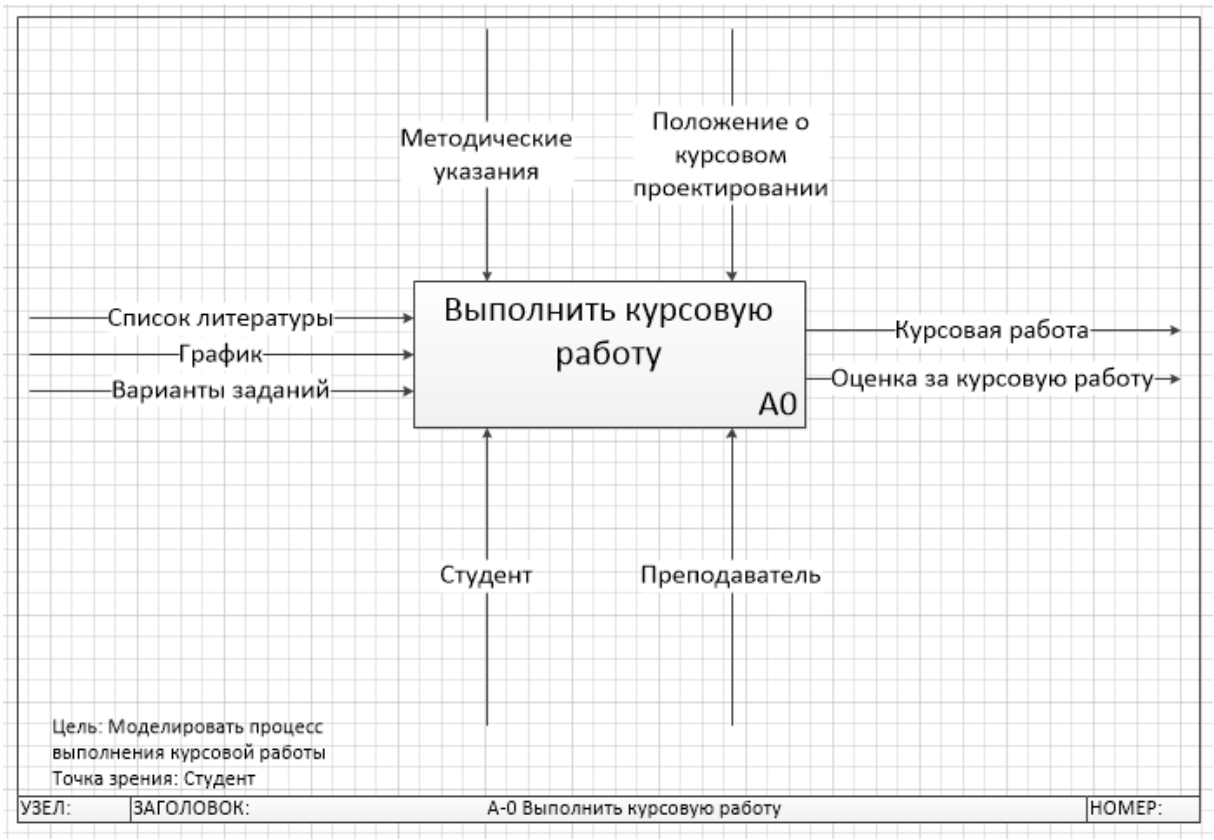


Рисунок 26 – Контекстная диаграмма

4.2. Создание диаграммы декомпозиции

1. Для построения декомпозиции диаграммы создайте новую страницу путем нажатия правой кнопкой мыши в нижнем левом углу окна на ярлык Страница 1. Выбрать пункт Добавить страницу (рис. 8)

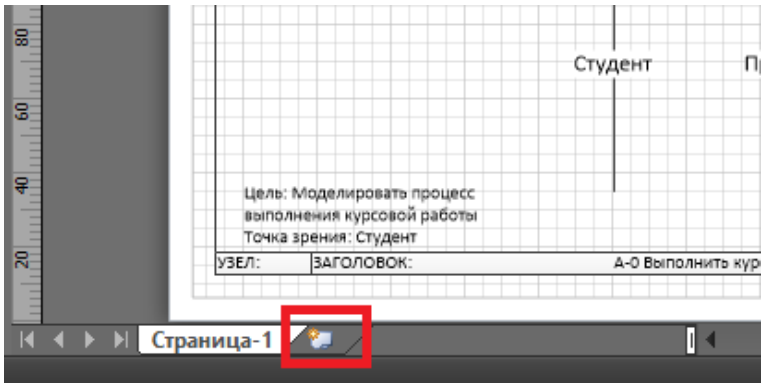


Рисунок 27 – Добавление страницы

2. Переименуйте страницы в соответствии с уровнем декомпозиции, например: A-0, A1 и т.д.
3. Распределите работы диаграммы декомпозиции в области Блока заголовка в соответствии с табл. 2.

Таблица 6 – Работы диаграммы декомпозиции A0

Имя работы (Activity Name)	**Определение (Definition)**
Получить задание	Выбрать задание из списка, согласовать его с преподавателем
Подобрать литературу	Выбрать из списка литературы подходящие источники
Сделать расчеты	Выполнить (если необходимо) расчетную часть курсовой работы согласно заданию
Сделать графическую часть	При необходимости сделать графики и чертежи
Оформить пояснительную записку	Оформить текстовую часть и объединить все сделанные части в единое целое
Получить консультацию	Получить консультацию у преподавателя перед защитой, выявить неточности и недостатки
Защитить курсовую работу	Сдать готовую курсовую работу и ответить на вопросы преподавателя

4. Распределите стрелки для диаграммы декомпозиции в соответствии с контекстной диаграммой. Для этого «перенесите» входные и выходные стрелки, связанные с декомпозируемой работой, в поле декомпозиции.

Итог выполнения вышеописанных шагов представлен на рис. 9.

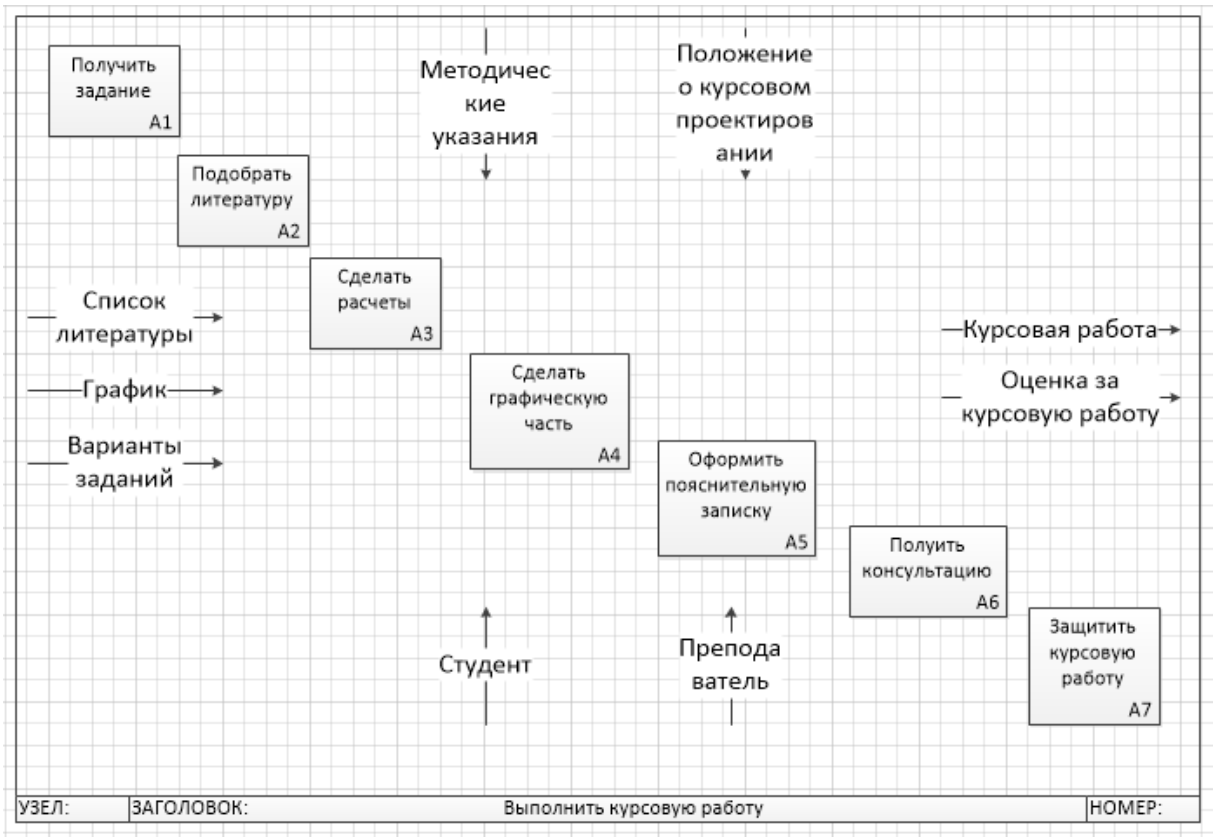


Рисунок 28 – Диаграмма декомпозиции

Разветвление стрелок. График (расписание) необходимо для того, чтобы прийти на консультацию и на защиту, т.е. необходимо подвести одноименную стрелку к 2 работам. Для разветвления стрелки необходимо от фрагмента стрелки до сегмента работы провести стрелку, состоящую из нескольких блоков Однонаправленное соединение.

Слияние стрелок. Для слияния двух стрелок выхода необходимо провести работы аналогичные разветвлению.

ICOM-метки. Используя блок текста, расставьте ICOM метки.

Результат выполнения предыдущих пунктов представлен на рисунке (рис. 9).

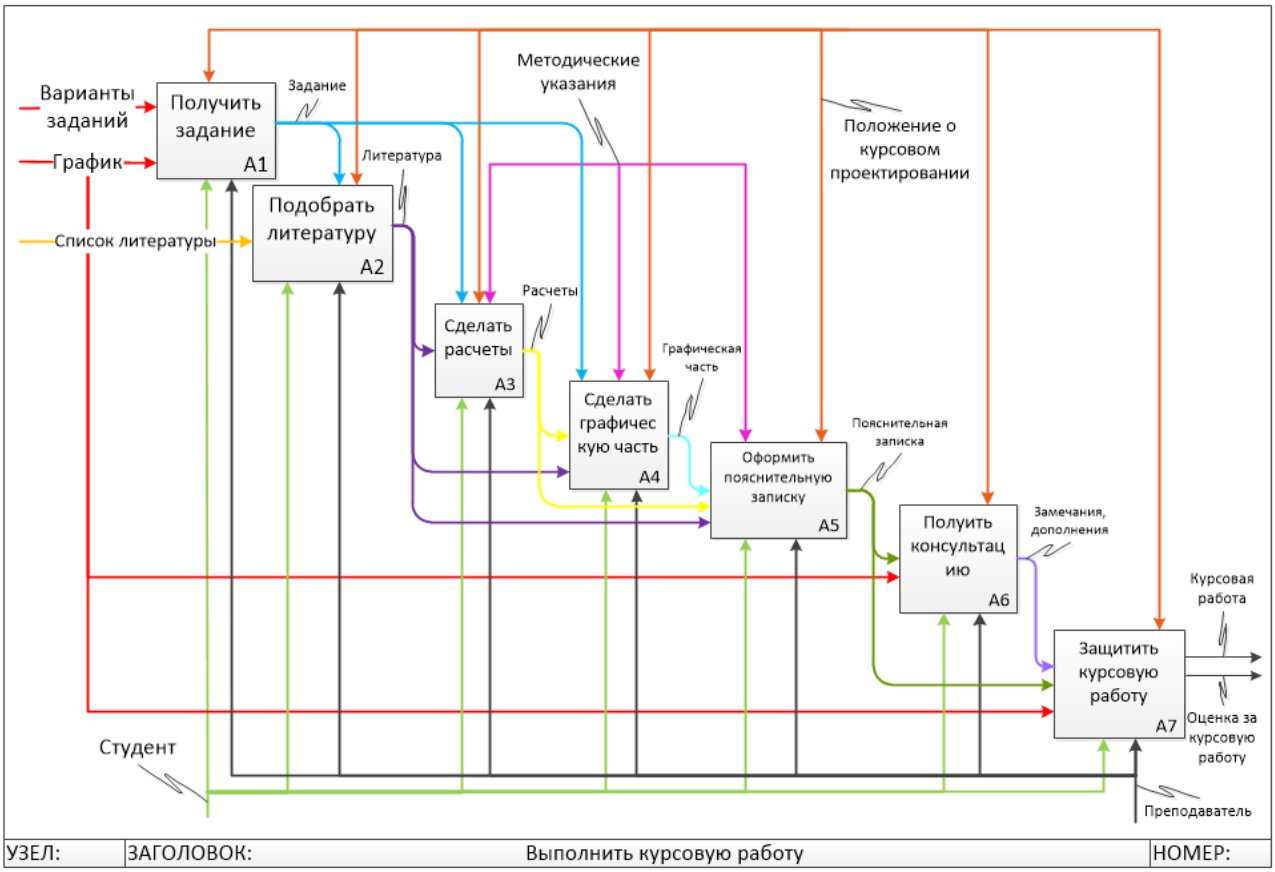


Рисунок 29 – Диаграмма декомпозиции блока A0

4.3. Создание дерева узлов

Дерево узлов – это диаграмма, отображающая иерархию работ процесса (рис. 11).

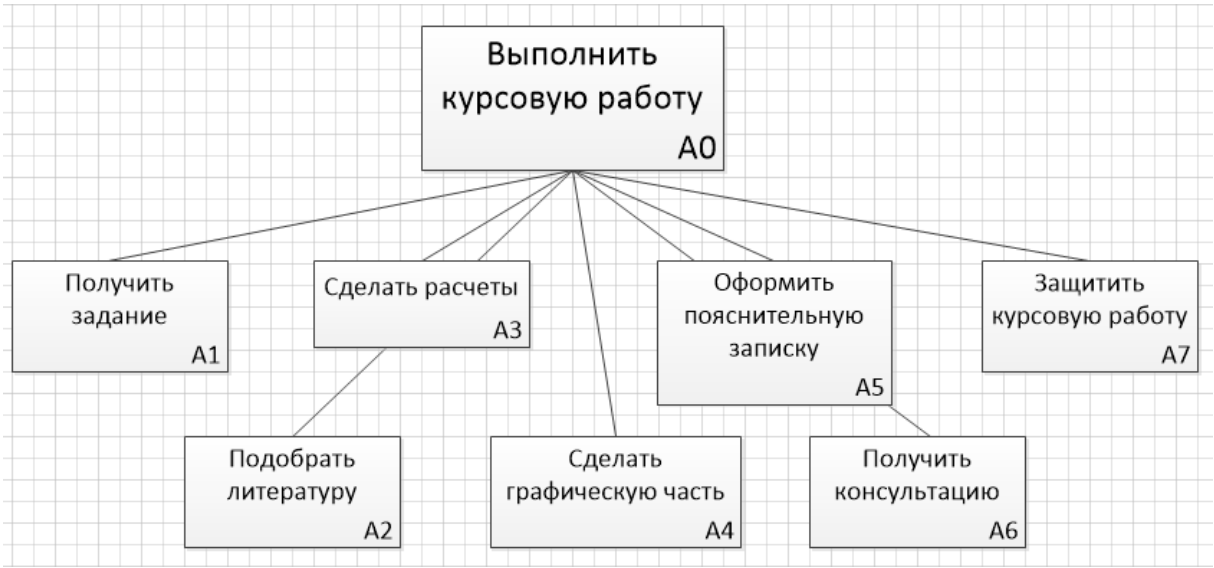


Рисунок 30 – Диаграмма узлов

Для построения диаграммы:

– создайте новую страницу;
– присвойте имя странице: дерево узлов;
– постройте дерево узлов, используя фигуры схемы IDEF0.

4.4. Создание глоссария

Глоссарий – это словарь ключевых слов, повествований, изложений, используемых при описании процесса (рис. 12, 13).

Для построения глоссария:

– создайте документ Microsoft Office Word;
– создайте 2 таблицы: описание работ процесса, описание интерфейсных дуг процесса;

– наименование столбцов таблиц: имя (работы/дуги, описание);
– заполните таблицы в соответствии с ранее разработанной моделью процесса.

Name	Definition
Выполнить курсовую работу	Текущие процессы выполнения курсовой работы
Защитить курсовую работу	Сдать готовую курсовую работу и ответить на вопросы преподавателя
Оформить пояснительную записку	Оформить текстовую часть и объединить все сделанные части в единое целое
Подобрать литературу	Выбрать из списка литературы подходящие
Получить задание	Выбрать задание из списка, согласовать его с
Получить консультацию	Получить консультацию у преподавателя перед
Сделать графическую часть	При необходимости сделать графики и чертежи
Сделать расчеты	Выполнить (если необходимо) расчетную часть курсовой

Рисунок 31 – Словарь работ

Name	Definition
Варианты заданий	Список заданий на курсовую работу, подлежа
График	График консультаций и сроки сдачи
Графическая часть	Выполненная графическая часть курсовой раб
Задание	Выдается на консультации преподавателем, чт
Замечания, дополнения	Замечания преподавателя, полученные на кон
Курсовая работа	Документ, являющийся основанием для получ
Литература	Выбранные источники, необходимые для выпо
Методические указания	Документ, содержащий указания по выполнен
Оценка за курсовую раб	Результат выполнения курсовой работы
Положение о курсовом п	Документ, отражающий организационные треб
Пояснительная записка	Теоретическая часть + расчеты + графическая
Преподаватель	Тот, кто оценивает курсовую работу
Расчеты	Выполненная расчетная часть курсовой работ
Список литературы	Источники информации для выполнения курсо
Студент	Тот, кто выполняет курсовую работу

Рисунок 32 – Словарь стрелок

5. Задание

На основе информационной модели предметной области, разработанной в практическом занятии №1, составить функциональную модель в нотации IDEF0.

Отчет по практическому занятию выполняется в формате MS Word, который содержит экранные формы моделей согласно заданию.

6. Варианты

11.Проектирование ИС «Отдел кадров»;
12.Проектирование ИС «Агентство аренды»;
13.Проектирование ИС «Аптека»;
14.Проектирование ИС «Ателье»;
15.Проектирование ИС «Аэропорт»;
16.Проектирование ИС «Библиотека»;
17.Проектирование ИС «Кинотеатр»;
18.Проектирование ИС «Поликлиника»;
19.Проектирование ИС «Автосалон»;
20.Проектирование ИС «Таксопарк».

7. Требования к построению модели

1. На контекстной диаграмме необходимо указать точку зрения и цель моделирования.
2. Количество блоков любой декомпозиции не менее 3-х и не более 9.
3. Количество декомпозиций – 3 уровня декомпозиции.

8. Контрольные вопросы

1. Каковы цели функционального моделирования?
2. Назовите основные компоненты функциональной модели.
3. Какие виды интерфейсных дуг различают в IDEF0?
4. Для чего нужна цель и точка зрения?
5. Что такое функциональный блок?
6. Какие виды диаграмм может содержать функциональная модель?

Практическая работа №3

Моделирование потоков данных в виде DFD-диаграмм с помощью MS Visio

1. Цель работы

Целью работы является изучение основных характеристик и основ работы с DFD-моделями в графическом редакторе Microsoft Visio.

2. Задачи

Основными задачами практической работы являются:

– изучение состава диаграмм DFD, назначения элементов каждого вида и способов их размещения на диаграмме;

– изучение операций по созданию DFD-модели в редакторе MS Visio.

3. Краткие теоретические сведения

3.1. Модель потоков данных

DFD – data flow diagrams – диаграммы потоков данных – методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ.

Диаграмма потоков данных – один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML.

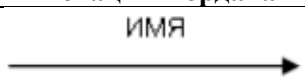
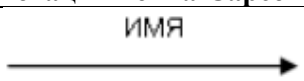
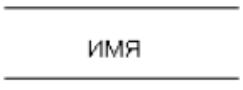
Для описания диаграмм DFD используются две нотации – Йордана (Yourdon) и Гейна-Сарсона (Gane-Sarson), отличающиеся синтаксисом.

3.2. Основные элементы информационной модели логического уровня

Согласно DFD источники информации (внешние сущности) порождают информационные потоки (потоки данных), переносящие информацию к подсистемам или процессам. Те в свою очередь преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям – потребителям информации.

При построении диаграмм различают элементы графической нотации, представленные в табл. 1.

Таблица 7 – Элементы графической нотации DFD

Наименование	Нотация Йордана	Нотация Гейна-Сарсона
Поток данных		
Процесс (система, подсистема)		
Накопитель данных		
Внешняя сущность		

Поток данных определяет информацию (материальный объект), передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т. д.

Каждый поток данных имеет имя, отражающее его содержание. Направление стрелки показывает направление потока данных. Иногда информация может двигаться в одном направлении, обрабатываться и возвращаться назад в ее источник. Такая ситуация может моделироваться либо двумя различными потоками, либо одним – двунаправленным.

На диаграммах IDEF0 потоки данных соответствуют входам и выходам, но в отличие от IDEF0 стрелки потоков на DFD могут отображаться входящими и выходящими из любой грани внешней сущности, процесса или накопителя данных.

Процесс (в IDEF0 – функция, работа) представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом.

Каждый процесс должен иметь имя в виде предложения с глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже, например:

- «Ввести сведения о клиентах»;
- «Рассчитать допускаемую скорость»;
- «Сформировать ведомость допускаемых скоростей».

Номер процесса служит для его идентификации и ставится с учетом декомпозиции. Вложенность процессов обозначается через точку.

Преобразование информации может показываться как с точки зрения процессов, так и с точки зрения систем и подсистем. Если вместо имени процесса «Рассчитать допускаемую скорость» написать «Подсистема расчета допускаемых скоростей», тогда этот блок на диаграмме стоит рассматривать, как подсистему.

Накопитель (хранилище) данных представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Накопитель данных может быть реализован физически в виде ящика в картотеке, области в оперативной памяти, файла на магнитном носителе и т.д.

Накопителю обязательно должно даваться уникальное имя и номер в пределах всей модели (всего набора диаграмм). Имя накопителя выбирается из соображения наибольшей информативности для разработчика. Например, если в качестве накопителей выступают таблицы проектируемой базы данных, тогда в качестве имен накопителей рекомендуется использовать имена таблиц. Таким образом, накопитель данных может представлять собой всю базу данных целиком, совокупность таблиц или отдельную таблицу. Такое представление накопителей в дальнейшем облегчит построение информационной модели системы.

Внешняя сущность (терминатор) представляет собой материальный объект или физическое лицо, выступающие как источник или приемник информации (например, заказчики, персонал, программа, склад, инструкция). Внешние сущности на DFD по смыслу соответствуют управлению и механизмам, отображаемым на контекстной диаграмме IDEF0.

Определение некоторого объекта, субъекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ проектируемой информационной системы. В связи с этим внешние сущности, как правило, отображаются только на контекстной диаграмме DFD. В процессе анализа и проектирования некоторые внешние сущности могут быть перенесены на диаграммы декомпозиции, если это необходимо, или, наоборот, часть процессов (подсистем) может быть представлена как внешняя сущность.

4. Рекомендации по выполнению практического занятия

На основе модели предметной области, разработанной в практических занятиях №1 и 2, составить функциональную модель в нотации DFD.

Отчет по практическому занятию выполняются в формате MS Word, который содержит экранные формы моделей согласно заданию.

5. Методика выполнения практического занятия

В качестве примера рассматривается процесс выполнения студентом курсовой работы (курсового проекта).

5.1. Создание контекстной диаграммы

1. Запустите MS Visio.
2. На закладке выбора шаблона выберите категорию *Программы и БД* и в ней элемент *Схема модели потоков данных*. Нажмите кнопку *Создать* в правой части экрана.
3. Окно программы примет вид, подобный рис. 1.

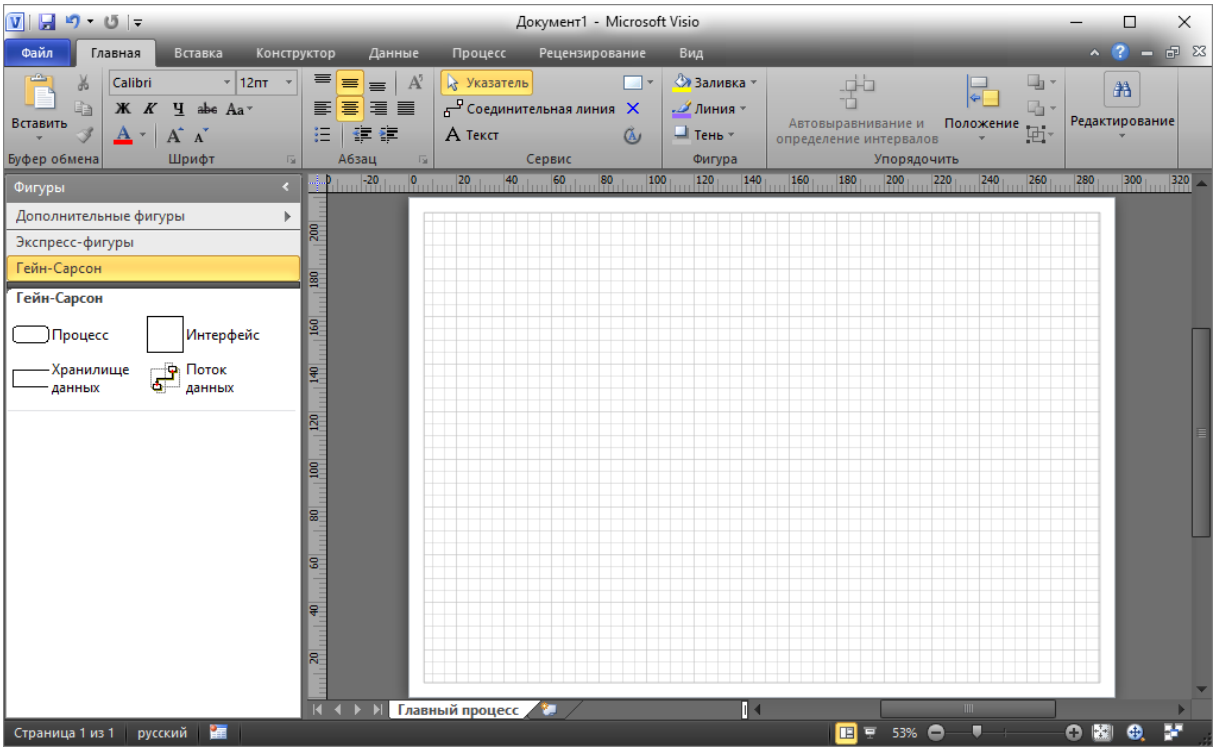


Рисунок 33 – Окно программы с выбранным шаблоном

4. Создание диаграммы.

4.1. Для удобства переведите страницу в альбомный вид: Конструктор – Параметры страницы – Ориентация – Альбомная;

4.2. Перетащите блок «*Процесс*» на пустую страницу, удерживая нажатой правую кнопку мыши;

4.3. Добавьте имя и номер процесса (рисунок 2), в данном случае: A0 Выполнить курсовую работу;

4.4. Далее, добавьте «*Внешние сущности (интерфейс)*» и потоки данных (рисунок 2).

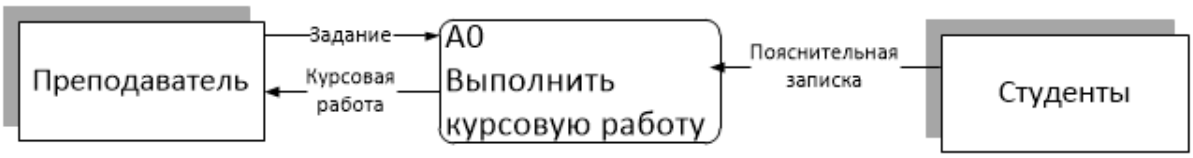


Рисунок 34 – Контекстная диаграмма в нотации DFD

5.2. Создание диаграммы декомпозиции

5. Для построения декомпозиции диаграммы создайте новую страницу (рис. 3)

Рисунок 35 – Добавление страницы

6. Переименуйте страницы в соответствии с уровнем декомпозиции, например: 1-й уровень, 2-й уровень и т.д.

7. Распределите процессы, интерфейсы и хранилища данных диаграмм декомпозиции на листе в соответствии с рис. 4 и 5.

8. Распределите стрелки для диаграммы декомпозиции в соответствии с рис. 4 и 5.

Рисунок 36 – Диаграмма декомпозиции 1-го уровня

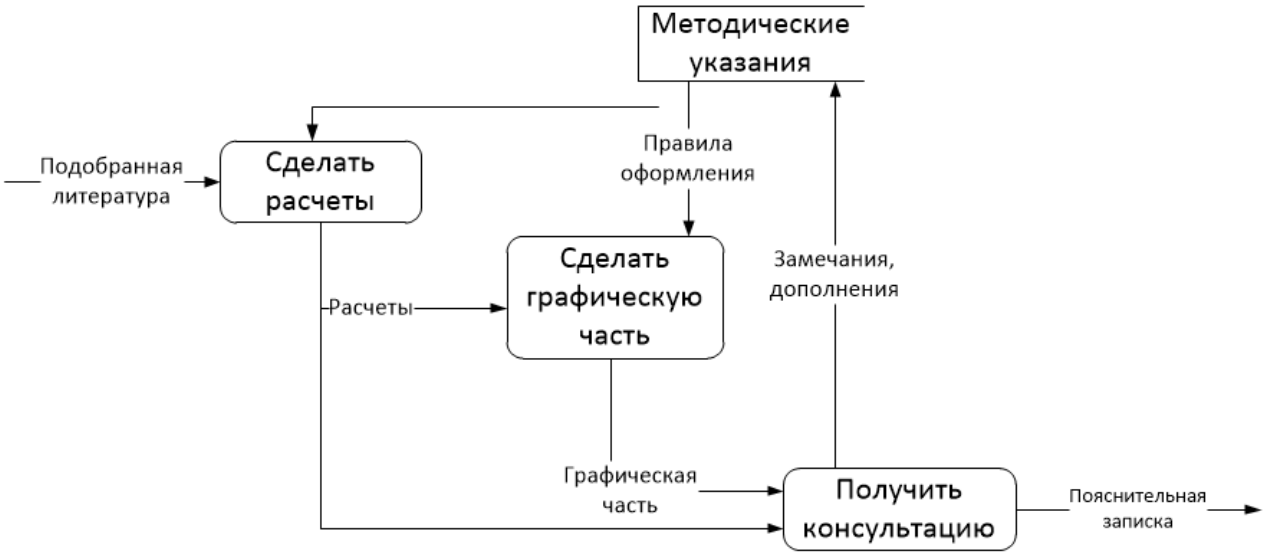


Рисунок 37 – Диаграмма декомпозиции 2-го уровня

6. Задание

На основе модели предметной области, разработанной в практических занятиях №1 и 2, составить функциональную модель в нотации DFD.

Отчет по практическому занятию выполняется в формате MS Word, который содержит экранные формы моделей согласно заданию.

7. Варианты

1. Проектирование ИС «Отдел кадров»;
2. Проектирование ИС «Агентство аренды»;
3. Проектирование ИС «Аптека»;
4. Проектирование ИС «Ателье»;
5. Проектирование ИС «Аэропорт»;
6. Проектирование ИС «Библиотека»;
7. Проектирование ИС «Кинотеатр»;
8. Проектирование ИС «Поликлиника»;
9. Проектирование ИС «Автосалон»;
10. Проектирование ИС «Таксопарк».

8. Требования к построению модели

Количество блоков любой декомпозиции не менее 3-х и не более 9.
Количество декомпозиций – 3 уровня декомпозиции.

9. Контрольные вопросы

1. Каково назначение стандарта DFD?
2. В чем основные отличия стандартов IDEF0 и DFD?
3. Каким образом в MS Visio создается схема DFD? Какие для этого используются нотации?
4. Какова роль основных элементов в стандарте DFD?

Практическая работа №4

Метод функционального моделирования IDEF0 с помощью Ramus Educational

1. Цель работы

Целью работы является изучение основ структурного подхода к проектированию ИС. Освоение принципов построения IDEF0-диаграммы классов в программной среде Ramus Educational.

2. Задачи

Основными задачами практической работы являются:

– ознакомиться с теоретическими вопросами структурного подхода к проектированию ИС;

– изучить диаграмму IDEF0 (Integration Definition for Function Modeling) для предметной области «Выполнение курсовой работы»;

– построить с помощью программного средства Ramus Educational диаграмму IDEF0 согласно индивидуальному заданию (вариант получить у преподавателя).

3. Краткие теоретические сведения

3.1. Общие положения структурного метода

Сущность структурного подхода к разработке ИС заключается в декомпозиции (разбиении) системы на автоматизируемые функции, которые в свою очередь делятся на подфункции, на задачи и так далее. Процесс декомпозиции продолжается вплоть до определения конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимосвязаны.

В основе структурного метода лежит несколько общих принципов:

– разбиение системы на множество независимых задач, доступных для понимания и решения;

– иерархическое упорядочивание, т.е. организация составных частей проблемы в древовидные структуры с добавлением новых деталей на каждом уровне.

К основным принципам относятся:

– абстрагирование, т.е. выделение существенных аспектов системы и отвлечение от несущественных;

– формализация, т.е. общее методологическое решение проблемы;

– непротиворечивость, состоящая в обосновании и согласовании элементов системы;

– иерархическая структуризация данных.

3.2. Метод функционального моделирования SADT

На основе метода SADT, предложенного Д. Россом, разработана методология IDEF0 (Icam DEFinition), которая является основной частью программы ICAM (Интеграция компьютерных и промышленных технологий), проводимой по инициативе ВВС США. Методология IDEF0 является наиболее признанным эффективным средством анализа, конструирования и отображения бизнес-процессов, применяемым также и широко за пределами США.

Метод SADT применяется при моделировании широкого круга систем, для которых определяются требования и функции, после чего проводится их реализация.

Методология SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели предметной области, которая отображает функциональную структуру, производимые функции и действия, а также связи между ними.

Результат применения метода SADT – модель, которая состоит из диаграмм, фрагментов текстов и глоссария со ссылками друг на друга. Все функции и интерфейсы представляются диаграммами в виде, соответственно, блоков и дуг. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке (исходные данные), указывается с левой стороны блока, а результаты работы функции (выход, результат) – с правой стороны. Механизм, осуществляющий операцию (человек или автоматизированная система), задается дугой, входящей в блок снизу (рис. 1).

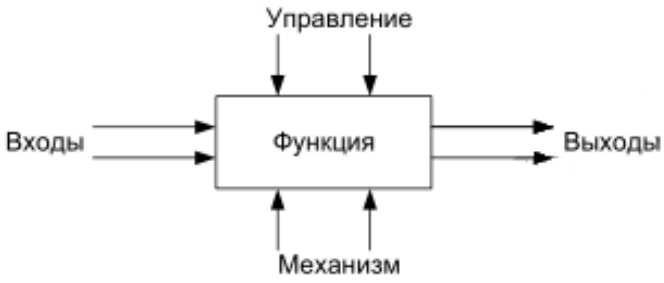


Рисунок 38 – Структура модели

Описание системы с помощью SADT называется **моделью. Субъектом** моделирования служит сама система. Однако моделируемая система никогда не существует изолированно: она всегда связана с окружающей средой. По этой причине в методологии SADT подчеркивается необходимость точного определения границ системы, т.е. модель устанавливает точно, что является и что не является субъектом моделирования, описывая то, что входит в систему, и, подразумевая то, что лежит за ее пределами. SADT-модель должна иметь единственный субъект.

С определением модели тесно связана позиция (называемая **точкой зрения**), с которой наблюдается система и создается ее модель. «Точку зрения» лучше всего представлять себе как место (роль, должность) человека или объекта в рассматриваемой системе, на которое надо «встать», чтобы увидеть систему в действии и необходимой полноте. У конкретной модели может быть только одна точка зрения.

Обычно вопросы для SADT-модели формулируются на самом раннем этапе проектирования, при этом основная суть этих вопросов должна быть выражена в одной-двух фразах, которые становятся целью модели.

После того как определены субъект, цель и точка зрения модели, начинается первая интеграция процесса моделирования по методологии SADT. Субъект определяет, что включить в модель, а что исключить из нее. Точка зрения диктует автору модели выбор нужной информации о субъекте и форму ее представления. Цель становится критерием окончания моделирования. Конечным результатом этого процесса является набор тщательно взаимоувязанных описаний, начиная с описания самого верхнего уровня системы и заканчивая подробным описанием ее деталей или отдельных операций.

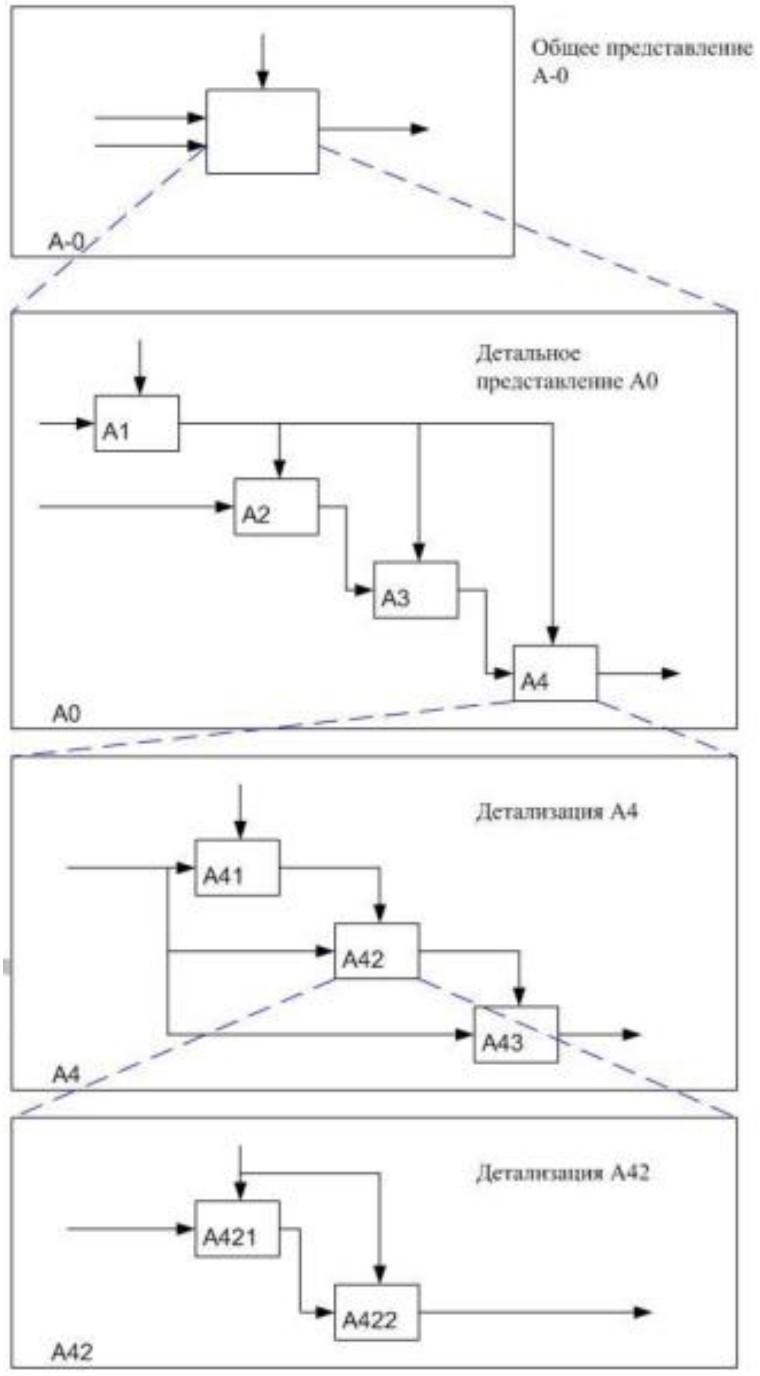


Рисунок 39 – Структура SADT-модели. Иерархия и декомпозиция диаграмм

Каждое из таких тщательно взаимосогласованных описаний называется диаграммой и имеет определенный уровень детализации. SADT-модель объединяет и организует диаграммы в иерархические структуры, в которых диаграммы наверху модели менее детализированы, чем диаграммы нижних уровней. Другими словами, модель SADT можно представить в виде древовидной структуры диаграмм, где верхняя диаграмма является наиболее общей, а самые нижние – максимально детализированы (рис. 2).

Каждый блок на диаграмме имеет свой номер. Блок любой диаграммы может быть детализирован диаграммой нижнего уровня, которая, в свою очередь, также может детализироваться с помощью необходимого числа диаграмм. Таким образом, формируется иерархия диаграмм. Для того чтобы указать положение любой диаграммы или блока в иерархии, им присваивают уникальные обозначения. Например, **A41** (А сокр. от Activity) является диаграммой, которая детализирует блок 1 на диаграмме А4. Аналогично, А4 детализирует блок 4 на диаграмме А0, которая является самой верхней (родительской) диаграммой модели.

Некоторые дуги имеют начало в одном из блоков диаграммы и завершение в другом, у других же начало может исходить от границ диаграммы – дуги управления, механизма, дуги входа и выхода, перенесенные с родительской (верхнего уровня) диаграммы. Таким образом, источник или получатель этих пограничных дуг может быть обнаружен только на родительской диаграмме.

Также следует сказать о так называемых «туннельных дугах». Туннельные дуги означают, что данные, выраженные этими дугами не рассматриваются на следующем уровне детализации (как бы проходят «насквозь»). Если «туннель» расположен в месте соединения дуги с блоком « », то данные этой дуги не обязательны на следующем уровне детализации. Если же «туннель» находится на противоположном конце дуги « » – это значит, что данные дуги не описываются на родительской диаграмме. Граничные дуги должны продолжаться (дублироваться) на родительской диаграмме, делая ее полной и непротиворечивой (рис. 3).

Для упрощения понимания приведенных диаграмм, следует расшифровать применяемую в IDEF систему обозначений, позволяющую аналитику точно идентифицировать и проверять по дугам связи между диаграммами. Эта схема кодирования дуг – «**ICOM**» – получила название по первым буквам английских эквивалентов слов вход (**Input**), управление (**Control**), выход (**Output**), механизм (**Mechanism**).

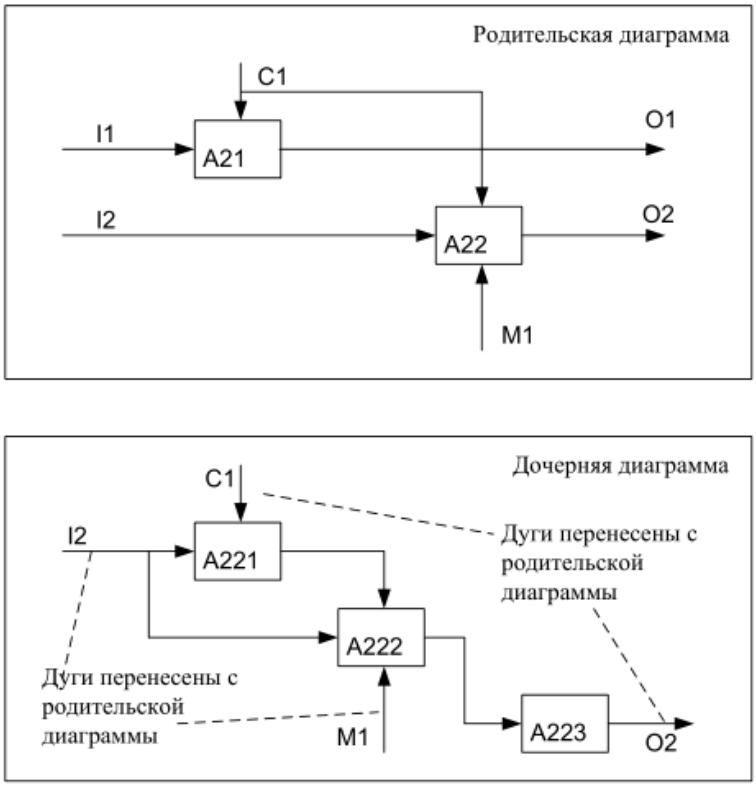


Рисунок 40 – Соответствие дуг родительской и дочерних диаграмм

4. Методика выполнения лабораторной работы

В качестве примера рассматривается процесс выполнения студентом курсовой работы (курсового проекта).

Программное обеспечение «Ramus» предназначено для использования в проектах, в которых необходимо описание бизнес-процессов предприятия. «Ramus» поддерживает методологии моделирования бизнес-процессов IDEF0 и DFD, а также имеет ряд дополнительных возможностей, призванных удовлетворить потребности команд разработчиков систем управления предприятиями.

«Ramus» обладает гибкими возможностями построения отчетности по графическим моделям, позволяющие создавать отчеты в форме документов, регламентирующих деятельность предприятия.

Ramus Educational имеет достаточно интуитивный интерфейс пользователя, позволяющий быстро и просто создавать сложные модели.

4.1 Начало работы

1. Запустите программу Ramus Educational. В появившемся окне (рис. 4) предлагается создать новый проект или открыть уже существующий.

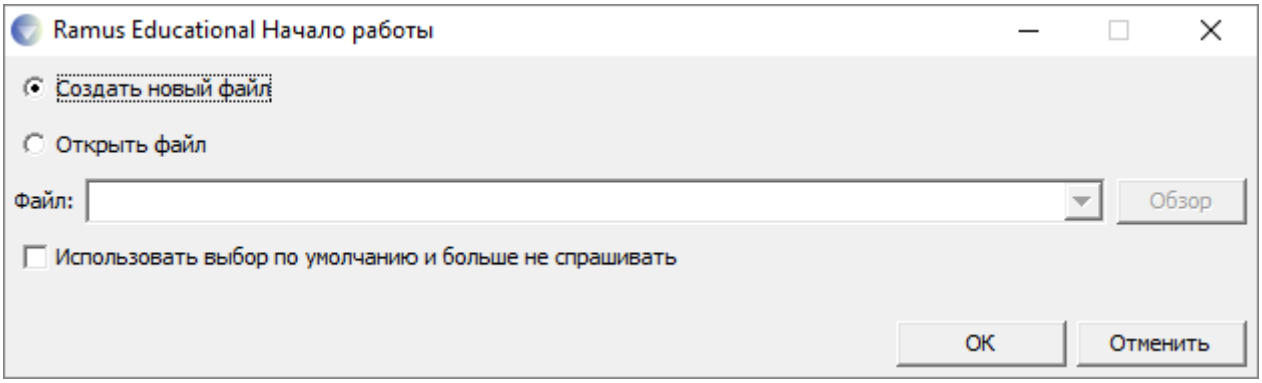


Рисунок 41 – Окно запуска Ramus Educational

2. После нажатия на кнопку «ОК» осуществляется запуск мастера проекта.

– На первом шаге (рис. 5) в соответствующие поля необходимо внести сведения об авторе, названии проекта и модели, а также выбрать тип нотации модели (IDEF0 или DFD).

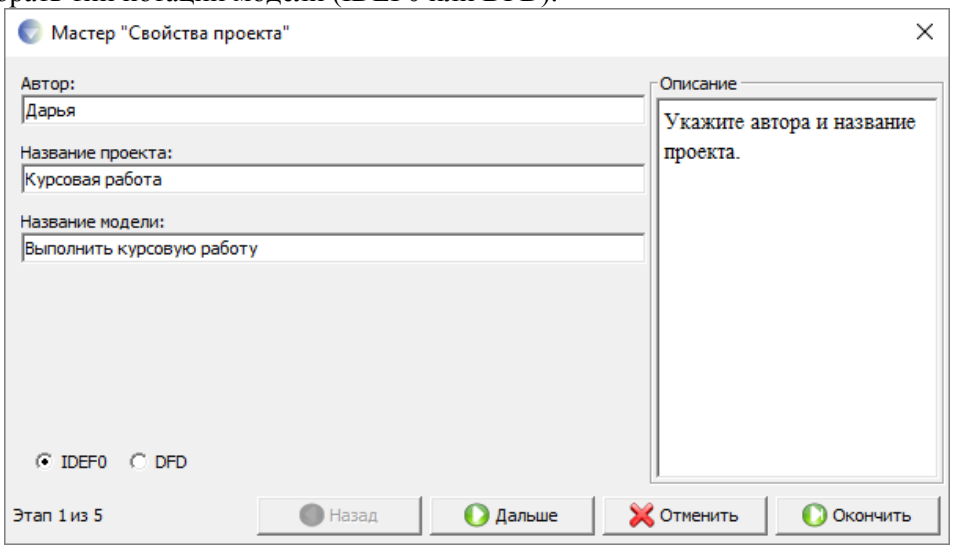


Рисунок 42 – Окно мастера создания проекта. Шаг 1

– На втором шаге вводится название организации, использующей данный проект.

– На третьем – дается краткое описание будущего проекта.

– Четвертый шаг позволяет создать несколько основных классификаторов (в данном случае можно пропустить этот шаг). Так как модели процессов реальных предприятий могут содержать значительное количество объектов (документы, персонал, функции и т.д.), то в Ramus предусмотрена возможность упорядочено хранить информацию об этих объектах в виде системы классификаторов. Классификация объектов упрощает поиск и обработку информации об объектах модели, а также и об объектах непосредственно не представленных на диаграммах процессов, но относящихся к процессам предприятия.

– На пятом, заключительном, предлагается выбрать те из созданных классификаторов, элементы которых будут содержаться в перечне собственников процессов (пропустить данный шаг).

При необходимости можно завершить работу мастера, нажав кнопку «Окончить».

После завершения работы мастера, откроется рабочее пространство «Диаграммы», в котором можно приступить к рисованию графической модели (рис. 6). В верхней части приводятся сведения о проекте, введенные пользователем посредством мастера диаграмм.

Программа Ramus Educational обладает гибким графическим интерфейсом, который можно настроить под нужды и предпочтения конкретного пользователя: ненужные окна можно закрыть/свернуть; можно менять их размеры и месторасположение; также можно группировать два и более окон в одном, при этом содержимое вложенных окон будет размещено на вкладках общего окна (данный функционал возможен не для всех комбинаций окон).

3. Сохраните созданную модель, выбрав опцию меню «Файл» – «Сохранить как».

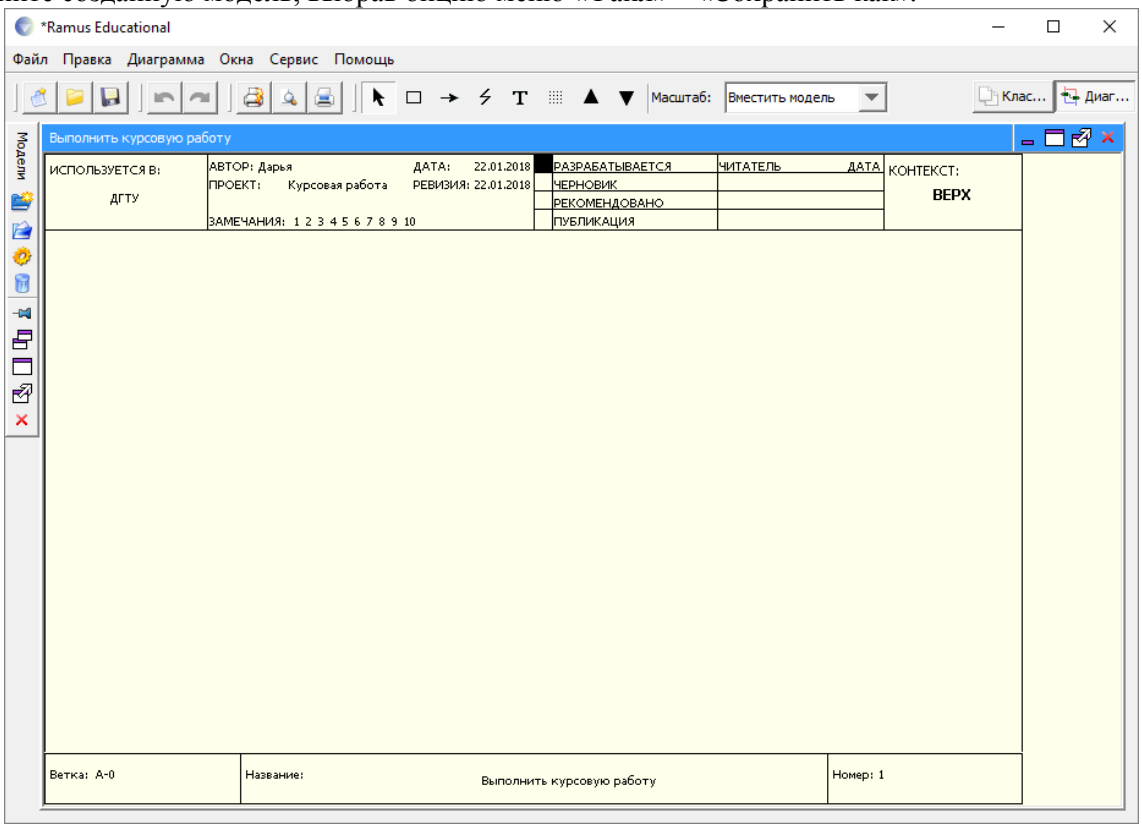


Рисунок 43 – Рабочее пространство

4.2 Создание контекстной диаграммы

1. На панели инструментов выберите пиктограмму функции ( ) и мышью укажите месторасположение на рабочем пространстве.

1. Дайте данному функциональному блоку имя «Выполнить курсовую работу». Для этого дважды щелкните внутри блока.

2. Используя пиктограмму панели инструментов , создайте стрелки на контекстной диаграмме согласно Таблица 1.

Таблица 8 – Контекстная диаграмма

Имя стрелки (Arrow Name)	**Определение стрелки (Arrow Definition)**	**Тип стрелки (Arrow Type)**
График	График консультаций и сроки сдачи	Input
Список литературы	Источники информации для выполнения курсовой работы	Input
Варианты заданий	Список заданий на курсовую работу, подлежащий распределению между студентами	Input
Методические указания	Документ, содержащий указания по выполнению курсовой работы, описывающий содержание ее частей и основные требования	Control
Положение о курсовом проектировании	Документ, отражающий организационные требования по выполнению и сдаче курсовой работы	Control
Курсовая работа	Документ, являющийся основанием для получения оценки	Output
Оценка за курсовую работу	Результат выполнения курсовой работы	Output
Студент	Тот, кто выполняет курсовую работу	Mechanism

3. В результате должна получиться контекстная диаграмма, показанная на рис. 7.

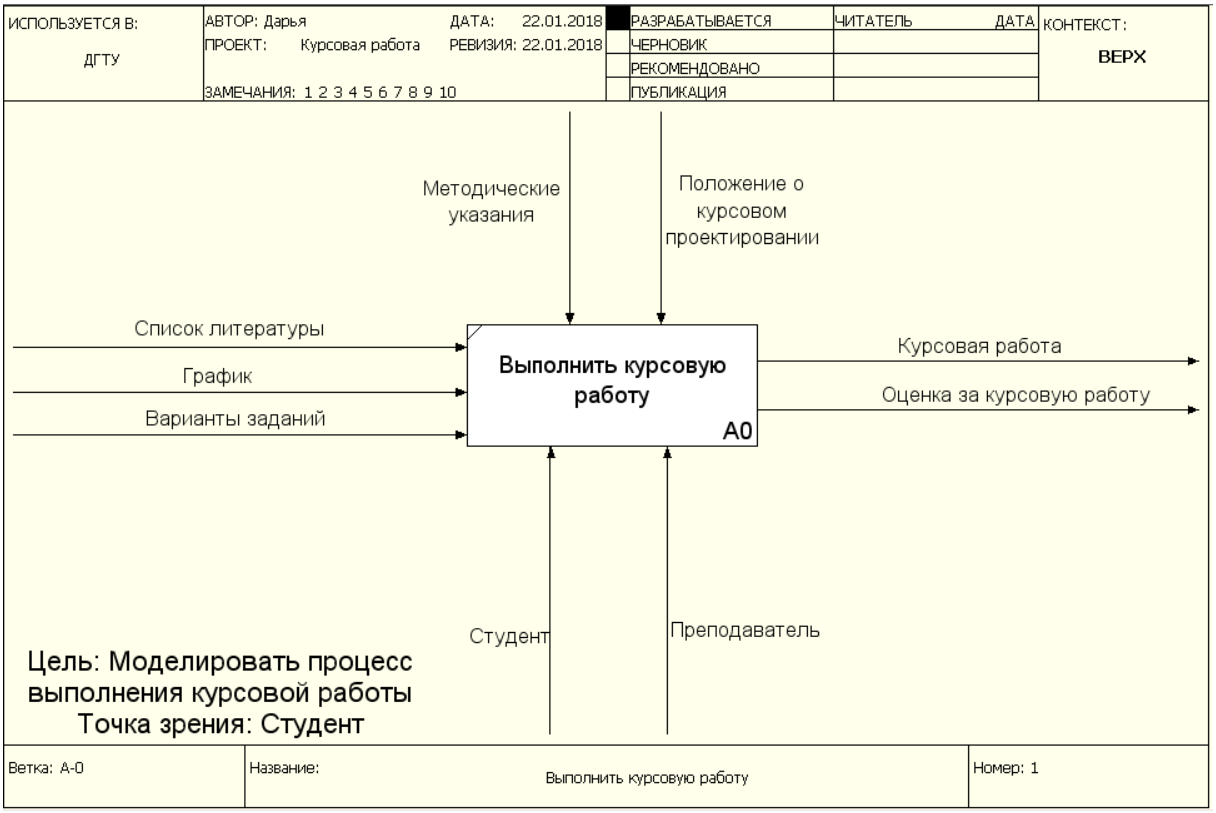


Рисунок 44 – Контекстная диаграмма предметной области «Выполнение курсовой работы»

4.3 Создание диаграммы декомпозиции

1. Выберите в палитре инструментов кнопку перехода на нижний уровень  , в диалоговом окне «Создание новой диаграммы» (рис. 8) установите количество функциональных блоков 7, укажите тип диаграммы (IDEF0) и нажмите кнопку ОК.

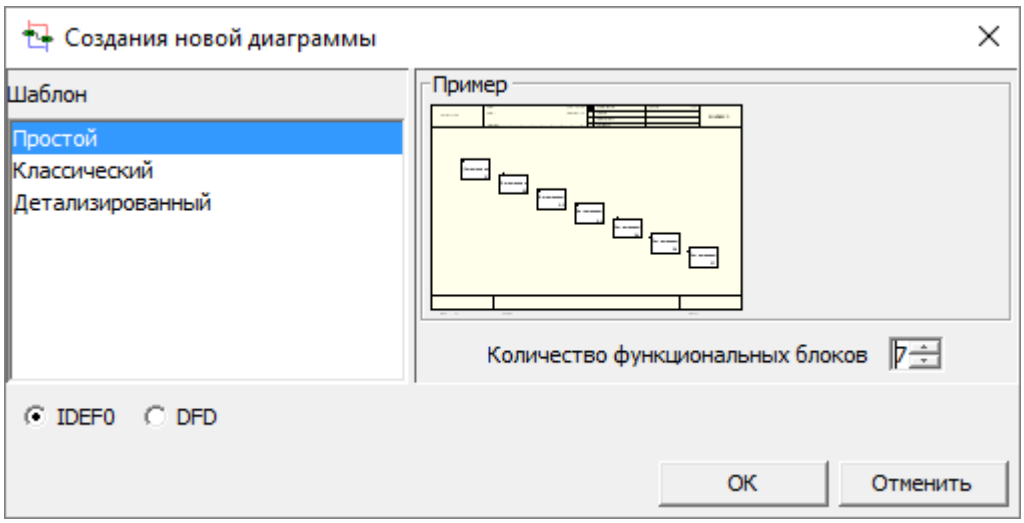


Рисунок 45 – Диалоговое окно создания детализирующей диаграммы

2. Автоматически будет создана диаграмма первого уровня декомпозиции (рис. 9) с перенесенными в нее потоками родительской диаграммы.

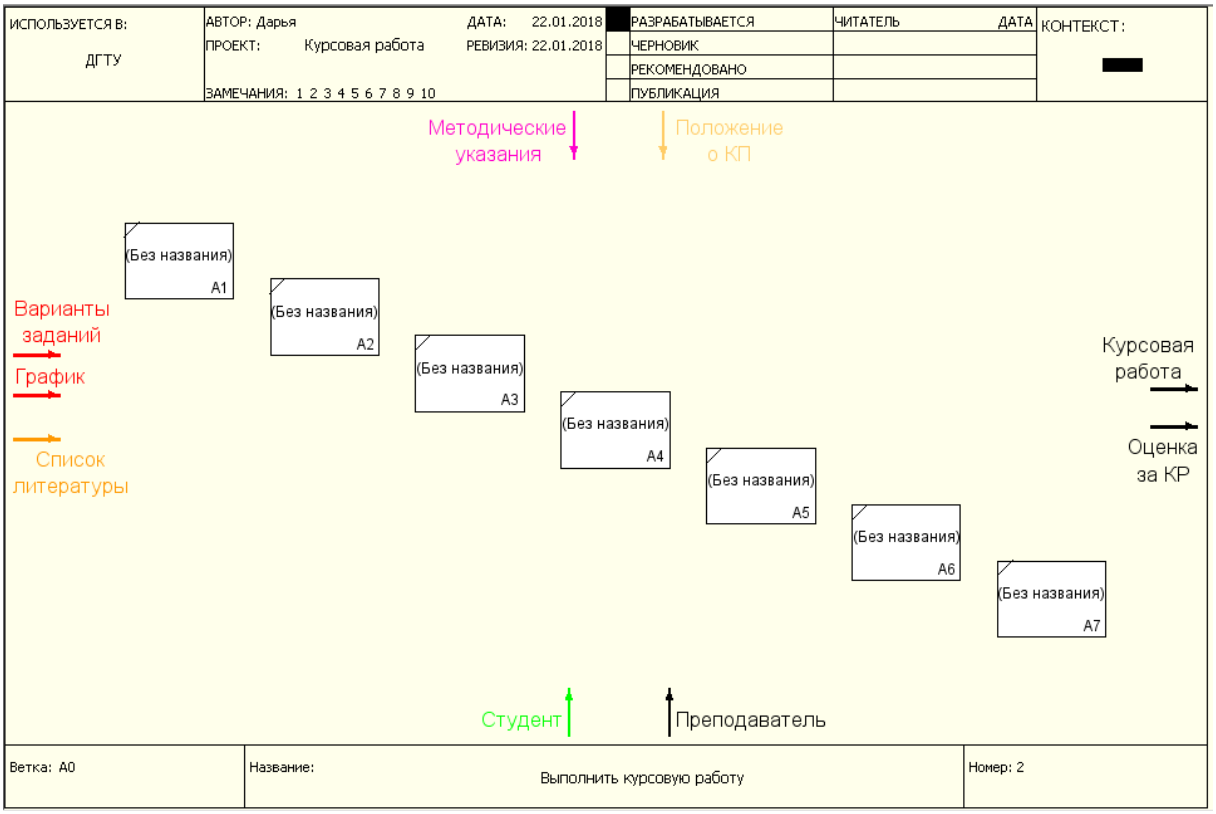


Рисунок 46 – Рабочее пространство детализирующей диаграммы

3. Выделите первую работу (функциональный блок), затем двойным щелчком мыши или, выбрав в контекстном меню пункт «Редактировать активный элемент», откройте окно свойств и внесите имя работы. Повторите операцию для оставшихся работ.

4. Выделив необходимый поток (стрелку) и, удерживая левую клавишу мыши, соедините его требуемым образом (через вход, управление, механизм или выход) с соответствующим функциональным блоком. В результате должна получиться детализирующая диаграмма, представленная на рис. 10.

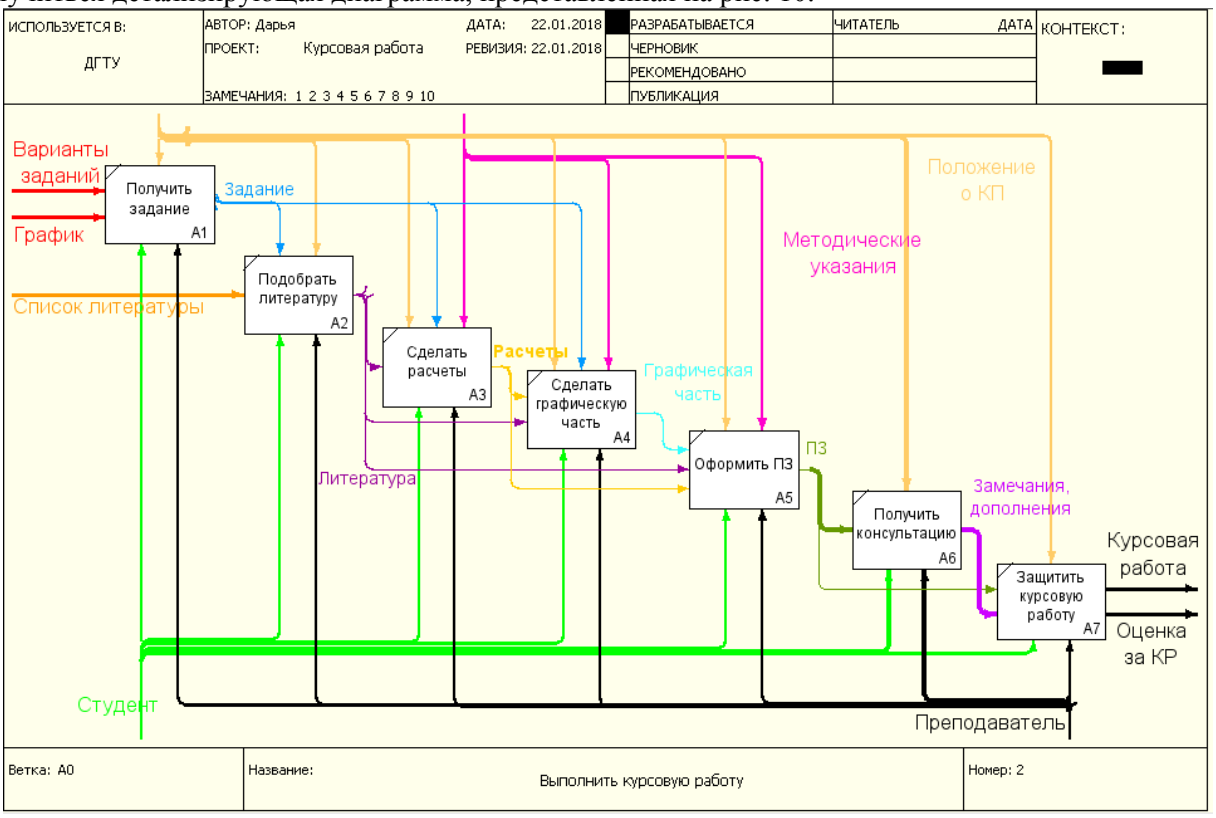


Рисунок 47 – Детализированная диаграмма первого уровня

5. Задание

На основе информационной модели предметной области, разработанной в практическом занятии №1, составить функциональную модель в нотации IDEF0.

6. Требования к построению модели

1. На контекстной диаграмме необходимо указать точку зрения и цель моделирования.
2. Количество блоков любой декомпозиции не менее 3-х и не более 9.
3. Количество декомпозиций – 3 уровня декомпозиции.

7. Контрольные вопросы

1. Каковы цели функционального моделирования?
2. Назовите основные компоненты функциональной модели.
3. Какие виды интерфейсных дуг различают в IDEF0?
4. Для чего нужна цель и точка зрения?
5. Что такое функциональный блок?
6. Какие виды диаграмм может содержать функциональная модель?

Практическая работа №5

Моделирование потоков данных (процессов) DFD с помощью Ramus Educational

1. Цель работы

Целью работы является изучение основ создания диаграммы потоков данных. Освоить принципы построения диаграммы потоков данных в Ramus Educational.

2. Задачи

Основными задачами практической работы являются:

– ознакомиться с теоретическими вопросами моделирования потоков данных и методами построения спецификации процессов;

– изучить DFD-диаграммы для предметной области «Выполнение курсовой работы»;

– построить с помощью программного средства Ramus Educational DFD-диаграммы согласно индивидуальному заданию (вариант получить у преподавателя).

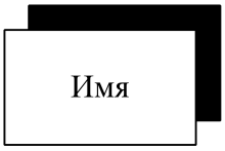
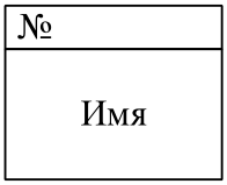
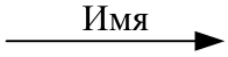
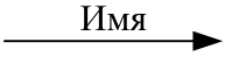
3. Краткие теоретические сведения

3.1. Общие сведения о диаграммах потоков данных

Диаграммы потоков данных (**DFD, Data Flow Diagrams**) являются основным средством моделирования функциональных требований проектируемой системы. С их помощью эти требования разбиваются на функциональные компоненты и представляются в виде сети связанных потоками данных процессов. Главная цель таких диаграмм – продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

Основные DFD-компоненты приведены в Таблице 1.

Таблица 9 – Основные элементы DFD-диаграммы

Название	Определение	Нотация Йордана	Нотация Гейна-Сарсона
Внешняя сущность	материальный предмет или физическое лицо, представляющее собой источник или приемник информации		
Процесс	преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом		
Поток	информация, передаваемая через некоторое соединение от источника к приемнику		
Хранилище	абстрактное устройство для хранения информации (жесткий диск, база данных и т.п.), которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми		

4. Методика выполнения лабораторной работы

В качестве примера рассматривается процесс выполнения студентом курсовой работы (курсового проекта).

Программное обеспечение «Ramus» предназначено для использования в проектах, в которых необходимо описание бизнес-процессов предприятия. «Ramus» поддерживает методологии моделирования бизнес-процессов IDEF0 и DFD, а также имеет ряд дополнительных возможностей, призванных удовлетворить потребности команд разработчиков систем управления предприятиями.

«Ramus» обладает гибкими возможностями построения отчетности по графическим моделям, позволяющие создавать отчеты в форме документов, регламентирующих деятельность предприятия.

Ramus Educational имеет достаточно интуитивный интерфейс пользователя, позволяющий быстро и просто создавать сложные модели.

4.1 Начало работы

4. Запустите программу Ramus Educational. В появившемся окне (рис. 1) предлагается создать новый проект или открыть уже существующий.

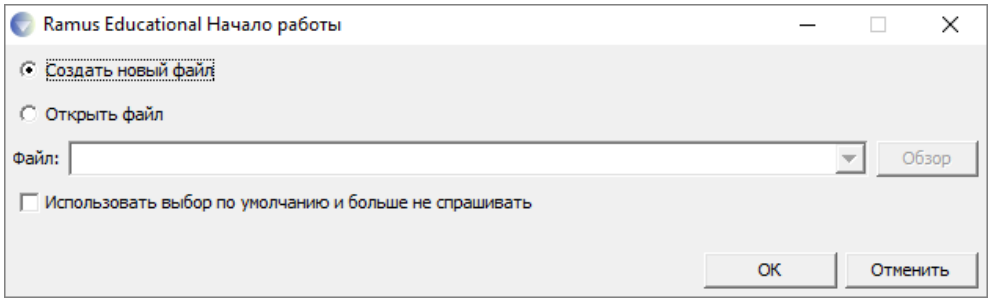


Рисунок 48 – Окно запуска Ramus Educational

5. После нажатия на кнопку «ОК» осуществляется запуск мастера проекта.

– На первом шаге (рис. 2) в соответствующие поля необходимо внести сведения об авторе, названии проекта и модели, а также выбрать тип нотации модели (IDEF0 или DFD) – в данном случае – DFD.

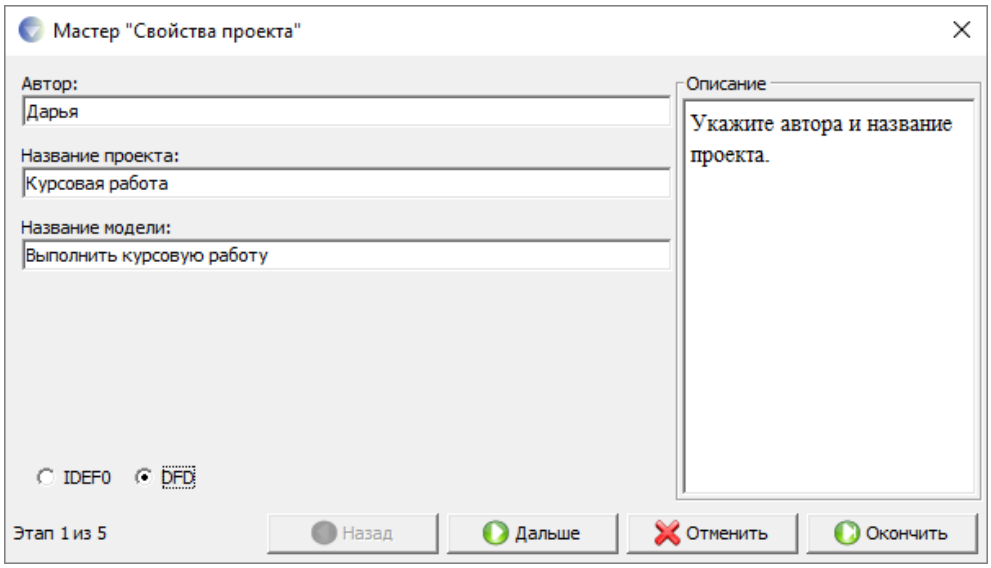


Рисунок 49 – Окно мастера создания проекта. Шаг 1

– На втором шаге вводится название организации, использующей данный проект.
– На третьем – дается краткое описание будущего проекта.
– Четвертый шаг позволяет создать несколько основных классификаторов (рисунок 3).

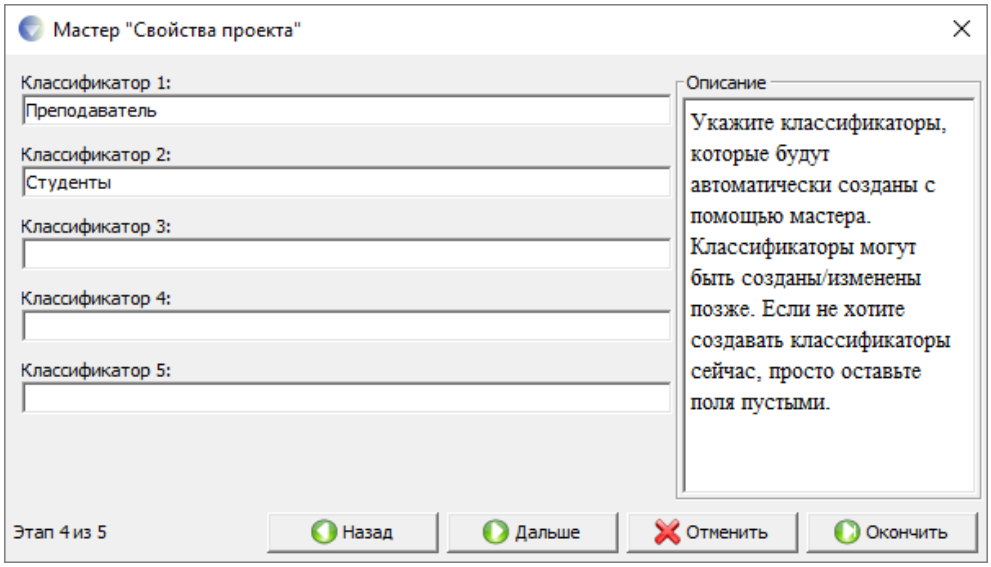


Рисунок 50 – Окно мастера создания проекта. Шаг 4

– На пятом, заключительном, предлагается выбрать те из созданных классификаторов, элементы которых будут содержаться в перечне собственников процессов (рис. 4).

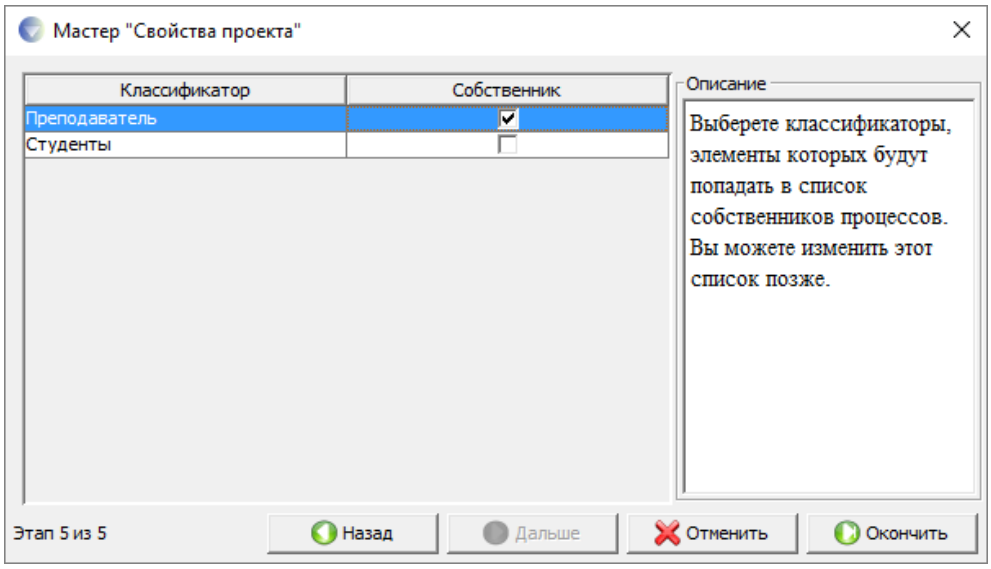


Рисунок 51 – Окно мастера создания проекта. Шаг 5

При необходимости можно завершить работу мастера, нажав кнопку «Окончить».

После завершения работы мастера, откроется рабочее пространство «Диаграммы», в котором можно приступить к построению графической модели. На панели инструментов, в верхней части окна рабочего пространства программы, содержатся элементы диаграммы потоков данных в нотации Gane-Sarson.

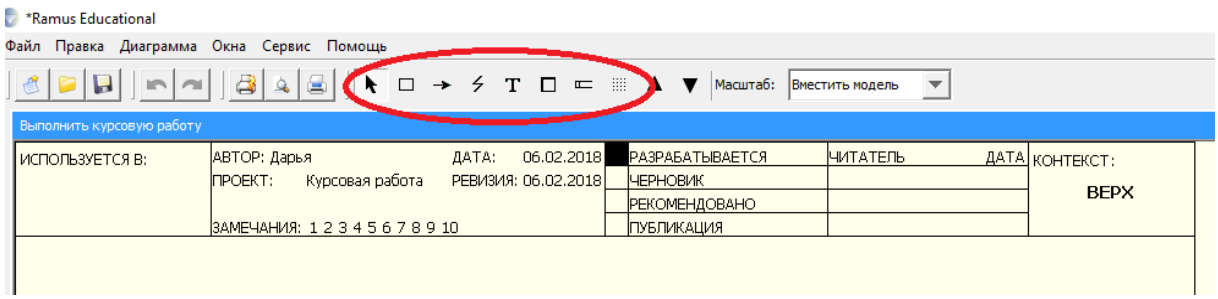


Рисунок 52 – Элементы DFD в нотации Гейна-Сарсона

6. Сохраните созданную модель, выбрав опцию меню «Файл» – «Сохранить как».

4.2 Создание контекстной диаграммы

2. На панели инструментов выберите инструмент создания процесса ( ) и мышью укажите месторасположение на рабочем пространстве нового процесса.

3. Выделив процесс, выберите в контекстном меню опцию **«Редактировать активный элемент»**. В появившемся диалоговом окне на вкладке **«Название»** присвойте процессу имя «*Выполнить курсовую работу*»; на вкладке «**Тип функционального блока**» укажите тип элемента – «Процесс» (Рис. 6).

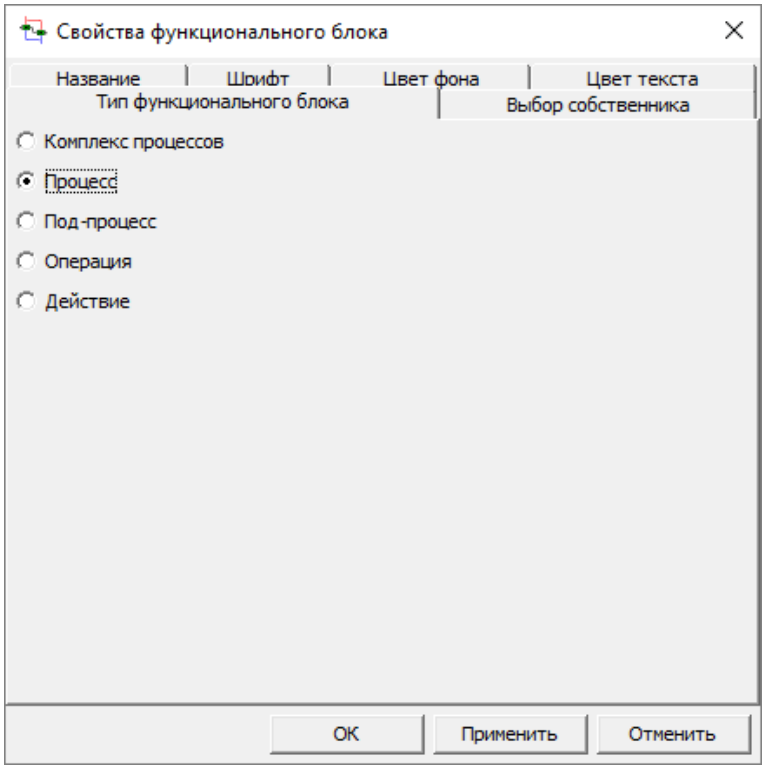


Рисунок 53 – Окно «Свойства функционального блока

4. На панели инструментов выберите инструмент создания внешней сущности  и мышью укажите произвольное ее месторасположение в области построения.

5. В контекстном меню созданной внешней сущности выберите опцию **«Редактировать активный элемент»**, на вкладке **«Объект»** нажмите **«Задать DFD объект»**, после чего, в появившемся окне выделите классификатор *«Преподаватель»* (см. Рис. 7-8) и нажмите «ОК».

6. Повторяя действия предыдущего шага, добавьте внешнюю сущность *«Студенты»*.

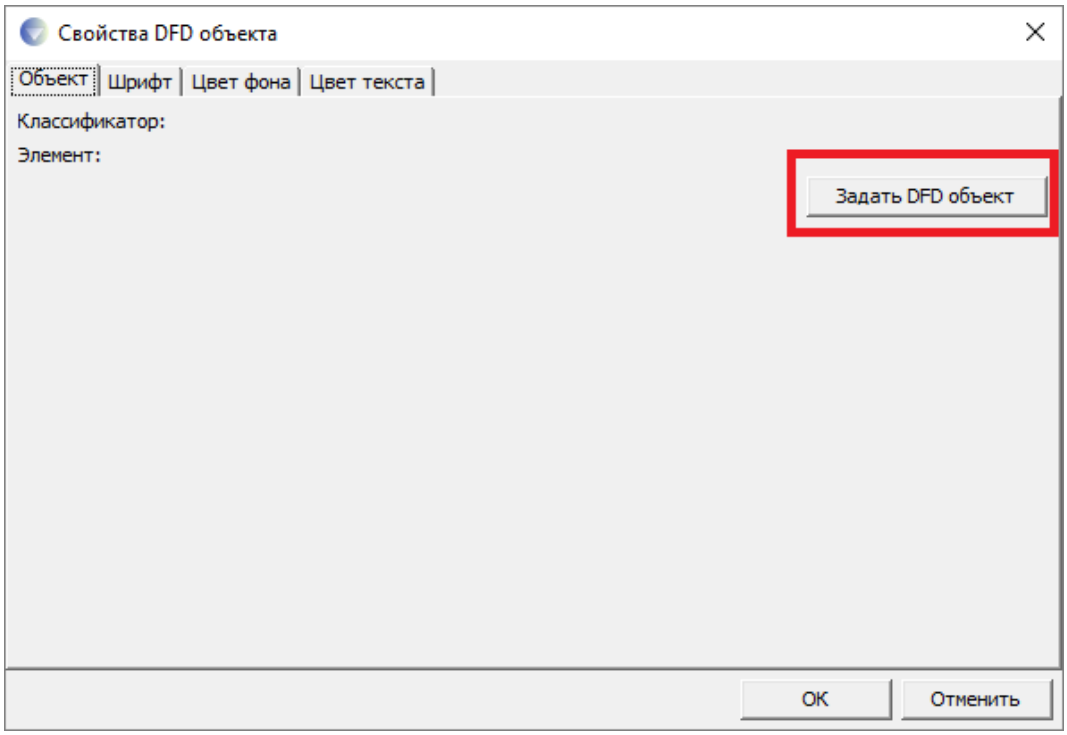


Рисунок 54 – Окно «Свойства DFD объекта»

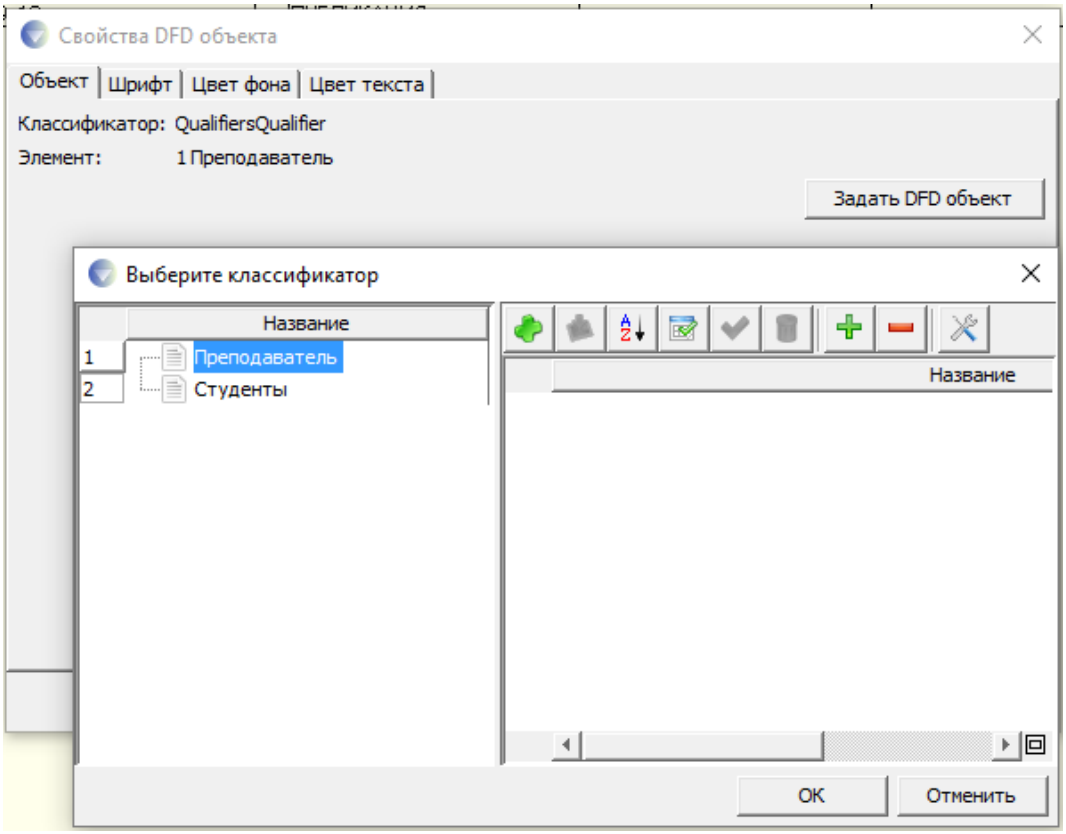


Рисунок 55 – Выбор классификатора

7. Выбрав на панели инструментов элемент →, создайте стрелки на контекстной диаграмме согласно Рис. 9. В результате должна получиться контекстная диаграмма, показанная на Рис. 9.

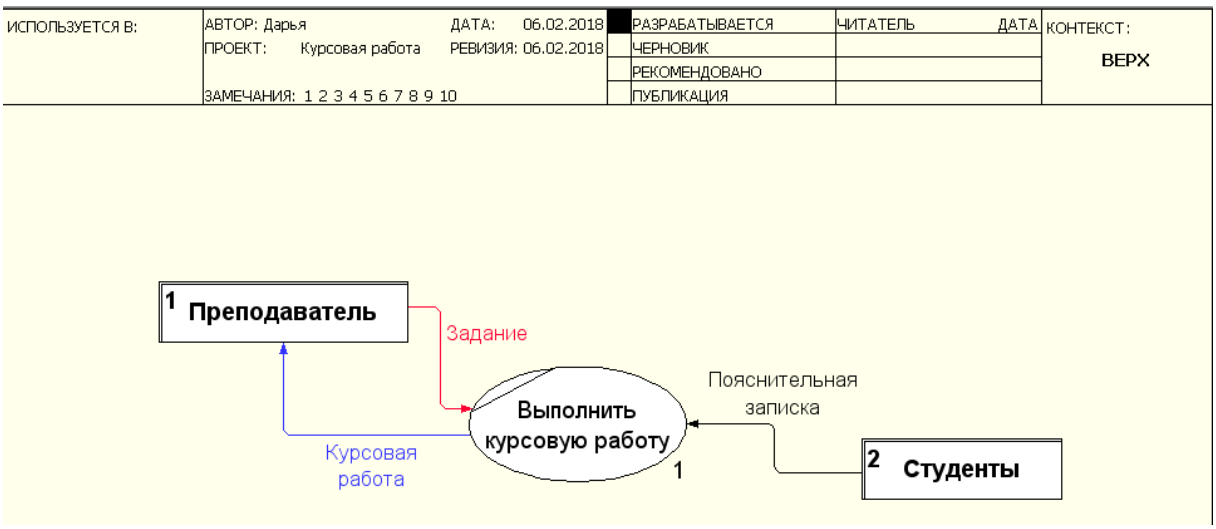


Рисунок 56 – Контекстная диаграмма предметной области «Выполнение курсовой работы»

4.3 Создание диаграммы декомпозиции

8. Выберите в палитре инструментов кнопку перехода на нижний уровень ▼, в диалоговом окне **«Создание новой диаграммы»** установите количество функциональных блоков 3, укажите тип диаграммы (DFD) и нажмите кнопку ОК.

9. Автоматически будет создана диаграмма первого уровня декомпозиции (рис. 10) с перенесенными в нее потоками родительской диаграммы.

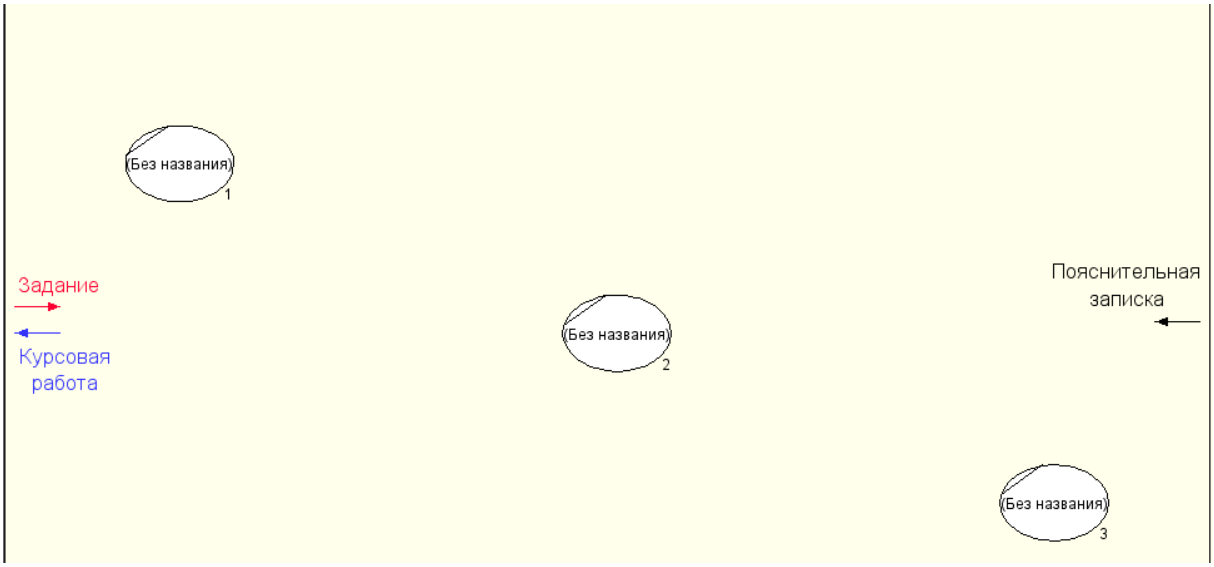


Рисунок 57 – Рабочее пространство детализирующей диаграммы

10.Двойным щелчком мыши на одном из процессов, или, выбрав в контекстном меню процесса пункт **«Редактировать активный элемент»**, откройте окно редактирования свойств и задайте процессу имя. Повторите операцию для оставшихся процессов.

11.Добавьте недостающие классификаторы для задания DFD объектов хранилищам данным. Для этого в меню выберите Окна -> Показать окно -> Классификаторы.

После этого нажмите на кнопку  . Название классификатора можно ввести в созданную строку, дважды, медленно кликнув мышью по строке, или же нажав клавишу F2, предварительно выделив нужную строку мышью.

12.Добавьте хранилища данных, воспользовавшись кнопкой палитры инструментов.

13.Выделив необходимый поток (стрелку) и, удерживая левую клавишу мыши, соедините его требуемым образом с соответствующим процессом. В результате должна получиться детализирующая диаграмма, представленная на Рис. 11.

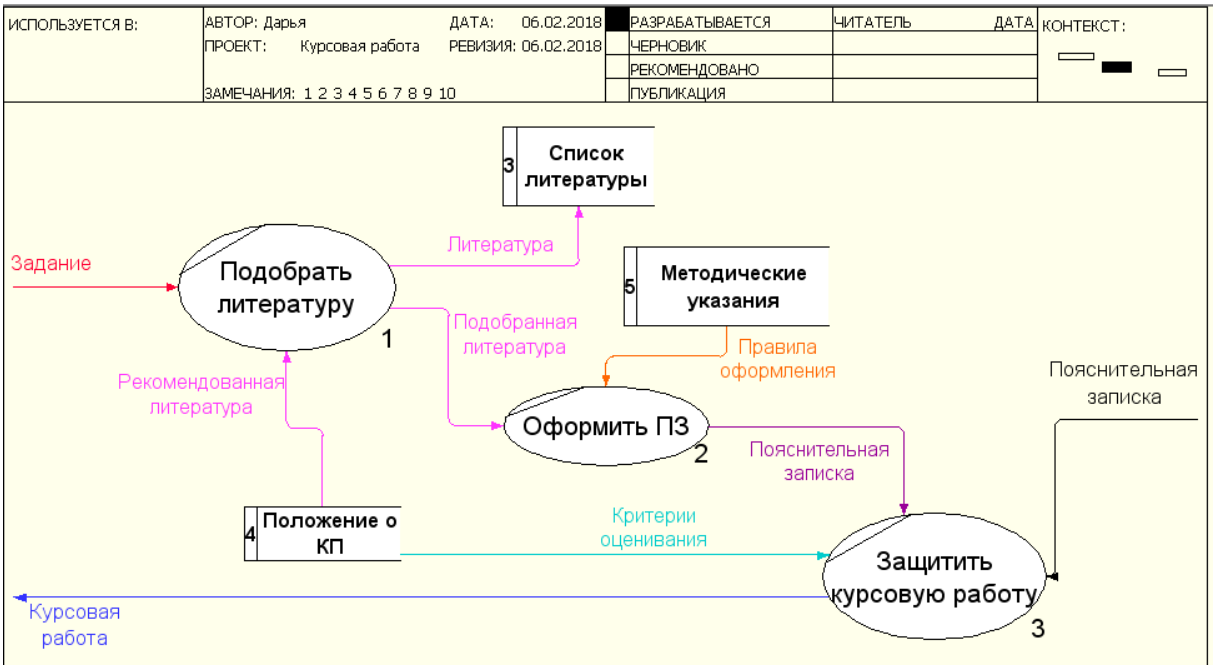


Рисунок 58 – Детализированная диаграмма первого уровня

14.На основе описанных выше действий постройте диаграмму декомпозиций второго уровня для процесса *«Оформить ПЗ»*. Построенная диаграмма должна иметь, изображенный на рисунке 13.

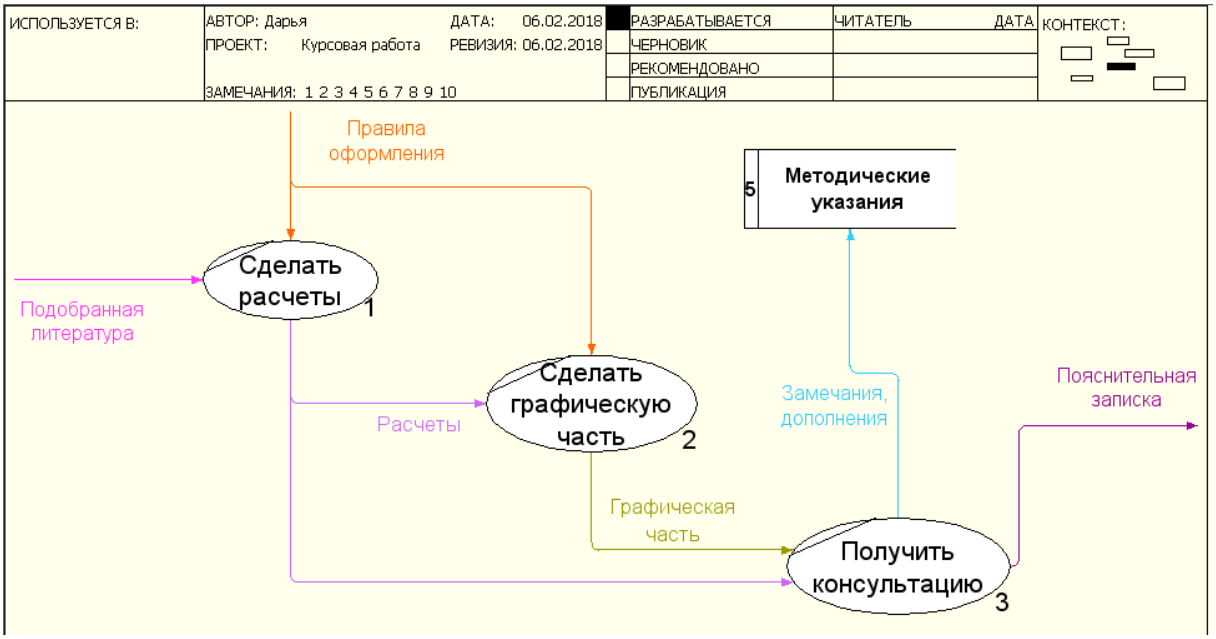


Рисунок 59 – Детализированная диаграмма второго уровня для процесса «Оформить ПЗ»

5. Задание

На основе модели предметной области, разработанной в практических занятиях №1 и 2, составить функциональную модель в нотации DFD.

Отчет по практическому занятию выполняется в формате MS Word, который содержит экранные формы моделей согласно заданию.

6. Варианты

1. Проектирование ИС «Отдел кадров»;
2. Проектирование ИС «Агентство аренды»;
3. Проектирование ИС «Аптека»;
4. Проектирование ИС «Ателье»;
5. Проектирование ИС «Аэропорт»;
6. Проектирование ИС «Библиотека»;
7. Проектирование ИС «Кинотеатр»;
8. Проектирование ИС «Поликлиника»;
9. Проектирование ИС «Автосалон»;
10. Проектирование ИС «Таксопарк».

7. Требования к построению модели

Количество блоков любой декомпозиции не менее 3-х и не более 9.
Количество декомпозиций – 3 уровня декомпозиции.

8. Контрольные вопросы

1. Каково назначение стандарта DFD?
2. В чем основные отличия стандартов IDEF0 и DFD?
3. Каким образом в MS Visio создается схема DFD? Какие для этого используются нотации?
4. Какова роль основных элементов в стандарте DFD?

Практическая работа №6

Построение организационных диаграмм с помощью MS Visio

1. Цель работы

Целью работы является изучение основ создания организационных диаграмм в ручном и автоматическом режимах.

2. Задачи

Основными задачами практической работы являются:

– ознакомиться с теоретическими вопросами построения организационных диаграмм с помощью MS Visio;
– ;
– .

3. Краткие теоретические сведения

3.1. Общие сведения об организационных диаграммах

Организационная диаграмма – это схема иерархии отчетности, которая используется для отображения отношений между сотрудниками, должностями и группами.

Организационные диаграммы могут быть как простыми схемами, так и большими и сложными на основе сведений из внешнего источника данных. Фигуры организационной диаграммы могут отображать основные сведения, например, имя и должность, или подробные сведения, например, отдел и учетный отдел. К фигурам организационной диаграммы можно даже добавлять рисунки.

Организационная диаграмма – это схематическое представление об иерархии внутри компании, а также распределении полномочий и ответственности между сотрудниками и отделами.

Как показывает опыт, применение руководителями, менеджерами данных схем значительно повышает эффективность управления предприятием.

Использование организационных диаграмм позволяет решить сразу несколько задач:

1. проанализировать состояние дел в компании на текущий момент,
2. вовремя обнаружить какую-то проблему в организации и системе управления,
3. избежать появления новых проблем и ошибок.

В зависимости от поставленных целей организационные диаграммы могут быть простыми или развернутыми.

Простые диаграммы содержат краткую информацию о каждом из сотрудников (имя, должность и место в системе организационной иерархии), а также раскрывают численный состав персонала.

Развернутые диаграммы несут дополнительные сведения о функциях и объектах управления сотрудников компании.

Организационные диаграммы позволяют визуально представить характер взаимоотношений внутри компании, благодаря чему удается своевременно обнаружить и разрешить возникающие в организации проблемы.

Основные **преимущества организационных диаграмм:**

1. Они позволяют выявить главных игроков в организации и проанализировать их отношения с остальным персоналом.

2. Сотрудники компании получают ясное представление о том, кто возьмет на себя ответственность при разрешении тех или иных организационных проблем в соответствии с корпоративными стандартами.

3. Повышают осведомленность сотрудников о функциях и профессиональном статусе каждого субъекта, работающего в компании, тем самым способствуя совершенствованию процесса коммуникации внутри фирмы.

Следует отметить, что организационные диаграммы имеют ряд ограничений. Например, отражая лишь структуру и формальный характер взаимодействия персонала, они не затрагивают личные взаимоотношения людей в коллективе. Кроме того, с помощью этих схем нельзя определить стиль менеджмента, выбранный руководством предприятия.

Организационные диаграммы описывают связи при помощи **трех типов графических элементов:**

1. ***Линия:*** указывает на прямые отношения между руководителями и подчиненными. Для описания взаимосвязей между различными иерархическими уровнями организации линии рисуют слева или справа от диаграммы.

2. ***Ступенька:*** служит для иллюстрации отношений между помощниками менеджера, а также связей между сферами деятельности, в которых помощники могут давать советы руководителю, но при этом их мнение не является авторитетным.

3. ***Функционал:*** показывает связи между должностями специалистов и сферами деятельности, в которых мнение специалистов является авторитетным для руководителя.

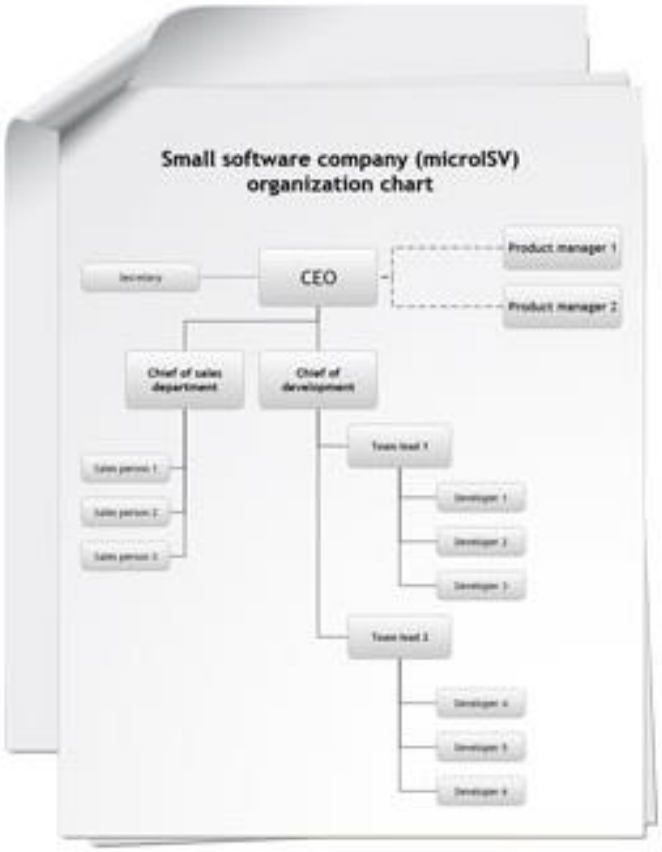


Рисунок 60 – Пример организационной диаграммы

Типы организационных структур

1. ***линейная модель:*** каждый руководитель обеспечивает руководство нижестоящими подразделениями по всем видам деятельности;
2. ***функциональная модель:*** «одно подразделение = одна функция»;
3. ***линейно-функциональная модель:*** ступенчатая иерархическая;
4. ***процессная модель:*** «одно подразделение = один процесс»;
5. ***матричная модель:*** «один процесс или один проект = группа сотрудников из разных функциональных подразделений»;
6. ***множественная (смешанная).***

В ***линейной структуре*** управления каждый руководитель обеспечивает руководство нижестоящими подразделениями по всем видам деятельности. *Достоинство* – простота, экономичность, предельное единоначалие. Основной *недостаток* – высокие требования к квалификации руководителей. Сейчас практически не используется.

Функциональная организационная структура – связь административного управления с осуществлением функционального управления.

Линейно-функциональная структура – линейные руководители являются единоначальниками, а им оказывают помощь функциональные органы. Линейные руководители низших ступеней административно не подчинены функциональным руководителям высших ступеней управления. Она применялась наиболее широко.

Преимущества:

– четкая система взаимных связей внутри функций и в соответствующих им подразделениях;

– четкая система единоначалия – один руководитель сосредотачивает в своих руках руководство всей совокупностью функций, составляющих деятельность;

– ясно выраженная ответственность;

– быстрая реакция исполнительных функциональных подразделений на прямые указания вышестоящих.

Недостатки:

– в работе руководителей практически всех уровней оперативные проблемы («текучка») доминируют над стратегическими;

– слабые горизонтальные связи между функциональными подразделениями порождают волокиту и перекладывание ответственности при решении проблем, требующих участия нескольких подразделений;

– малая гибкость и приспособляемость к изменению ситуации;

– критерии эффективности и качества работы подразделений и организации в целом разные и часто взаимоисключающие;

– большое число «этажей» или уровней управления между работниками, выпускающими продукцию, и лицом, принимающим решение;

– перегрузка управленцев верхнего уровня;

– повышенная зависимость результатов работы организации от квалификации, личных и деловых качеств высших управленцев.

Процессная модель. Истоки концепции управления процессами ведут к теориям управления, разработанным еще в XIX веке. В 80-х годах XIX-го века Фредерик Тейлор предложил менеджерам использовать методы процессного управления для наилучшей организации деятельности. В начале 1900-х годов Анри Файоль разработал концепцию реинжиниринга – осуществление деятельности в соответствии с поставленными задачами путем получения оптимального преимущества из всех доступных ресурсов.

Преимущества процессных структур:

– четкая система взаимных связей внутри процессов и в соответствующих им подразделениях;

– четкая система единоначалия – один руководитель сосредотачивает в своих руках руководство всей совокупностью операций и действий, направленных на достижение поставленной цели и получение заданного результата;

– наделение сотрудников большими полномочиями и увеличение роли каждого из них в работе компании приводит к значительному повышению их отдачи;

– быстрая реакция исполнительных процессных подразделений на изменение внешних условий;

– в работе руководителей стратегические проблемы доминируют над оперативными;

– критерии эффективности и качества работы подразделений и организации в целом согласованы и сонаправлены.

Недостатки процессной структуры:

– повышенная зависимость результатов работы организации от квалификации, личных и деловых качеств рядовых работников и исполнителей.

– управление смешанными в функциональном смысле рабочими командами – более сложная задача, нежели управление функциональными подразделениями;

– наличие в команде нескольких человек различной функциональной квалификации неизбежно приводит к некоторым задержкам и ошибкам, возникающим при передаче работы между членами команды. Однако потери здесь значительно меньше, чем при традиционной организации работ, когда исполнители подчиняются различным подразделениям компании.

Матричная модель. Матричные структуры совмещают принципы построения функциональных и процессных систем. В этих структурах существуют жестко регламентированные процессы, находящиеся под управлением менеджера процесса. При этом деятельность осуществляется работниками, находящимися в оперативном подчинении менеджера процесса и в административном подчинении руководителя в функциональном «колодце».

Преимущества

– Комплексный подход к реализации проекта, решению проблемы;

– Концентрация усилий на решении одной задачи, на выполнении одного конкретного проекта;

– Большая гибкость структуры;

– Активизация деятельности руководителей проектов и исполнителей в результате формирования проектных групп;

– Усиление личной ответственности конкретного руководителя как за проект в целом, так и за его элементы.

Недостатки

– При наличии нескольких организационных проектов или программ проектные структуры приводят к дроблению ресурсов и заметно усложняют поддержание и развитие производственного и научно-технического потенциала компании как единого целого;

– От руководителя проекта требуется не только управление всеми стадиями жизненного цикла проекта, но и учет места проекта в сети проектов данной компании;

– Формирование проектных групп, не являющихся устойчивыми образованиями, лишает работников осознания своего места в компании;

– При использовании проектной структуры возникают трудности с перспективным использованием специалистов в данной компании;

– Наблюдается частичное дублирование функций.

Смешанные структуры. Если применять различные модели организации деятельности в пределах отдельных бизнес-процессов, то можно использовать преимущества той или иной организационной модели. При этом для организации в целом будет применяться процессная организация основных структурных блоков, а в рамках отдельных блоков могут применяться различные модели.

4. Методика выполнения лабораторной работы

В качестве примера рассматривается процесс создания организационной диаграммы ВУЗа двумя способами:

1. вручную;

2. с помощью мастера.

4.1 Создание организационной диаграммы вручную

Для построения организационной диаграммы вручную необходимо проделать следующие действия:

1. Запустить *MS Visio*.
2. В разделе *Выберите шаблон* выбрать категорию *Бизнес*, а затем *Организационная диаграмма* и нажать кнопку *Создать*.

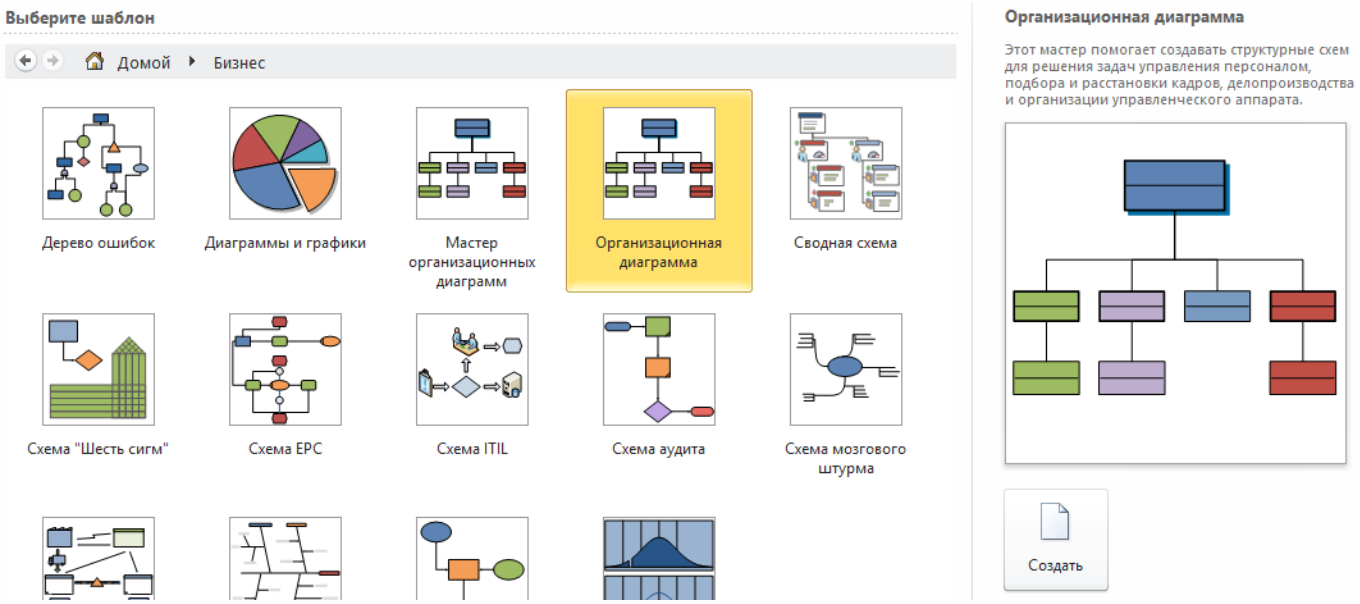


Рисунок 61 – Выбор шаблона

В разделе фигур можно увидеть основные элементы, необходимые для построения организационной диаграммы (рис. 3).

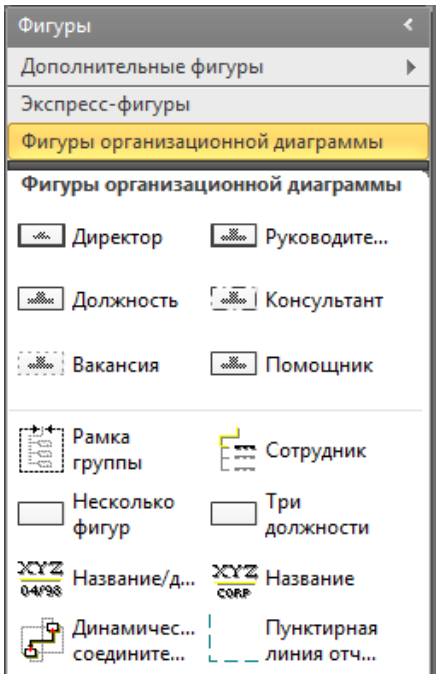


Рисунок 62 – Элементы организационной диаграммы в MS Visio

3. Перетащите фигуру *Директор* из набора фигур *Фигуры организационной диаграммы* в центр вверху страницы документа.
4. Надстройка организационных диаграмм отображает анимированное диалоговое окно (рис. 4), показывающее, как нужно добавлять в схему дополнительные фигуры.

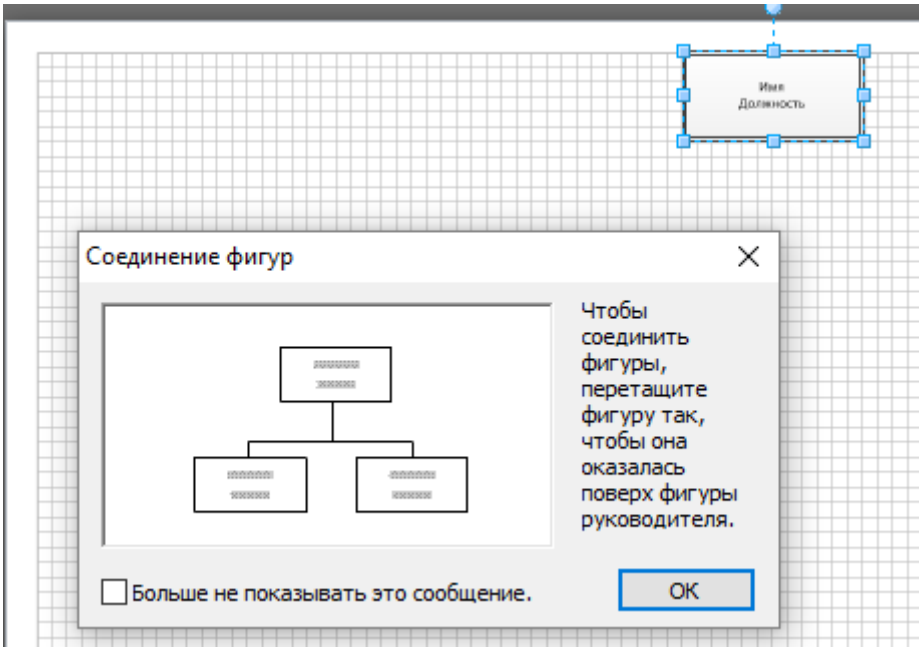


Рисунок 63 – Надстройка организационных диаграмм

5. Не снимая выделения с фигуры, при необходимости введите *ФИО*, затем нажмите клавишу ENTER и во второй строке введите *Должность*. (рис. 5).

При создании организационной диаграммы ВУЗа можно ограничиться только указанием должностей.

Рисунок 64 – Фигура Директор

6. Добавьте еще две фигуры *Директор* и расположите по обе стороны от фигуры *Ректор*. Соедините их используя *Соединительную линию.* В результате должна получиться схема, показанная на рисунке 6.

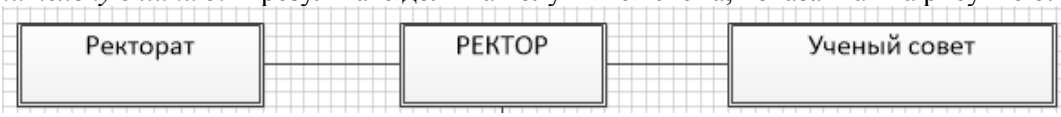


Рисунок 65 – Схема, полученная при выполнении шага 6

7. Перетащите фигуру *Руководитель* на фигуру *Директор*. Затем, при необходимости, введите ФИО, нажмите клавишу ENTER и введите *Должность*.

Надстройка автоматически размещает новую фигуру под фигурой *Директор*.

8. Повторите предыдущий шаг и обратите внимание, что надстройка разместила вторую фигуру руководителя сбоку от первой. Разместите все необходимые фигуры *Руководитель* в соответствии с рисунком 7.

Рисунок 66 – Фигуры Руководитель

9. Используя фигуры *Руководитель* и *Несколько фигур* добавьте необходимые *Должности,* относящиеся к руководителю **Первый проректор** (рис. 8)**.**

При использовании элемента *Несколько фигур* необходимо выполнить следующие действия:

– перетащить элемент Несколько фигур на необходимую фигуру;
– затем, в появившемся окне задать необходимые параметры, например, как на рисунке 9;
– нажать ОК.

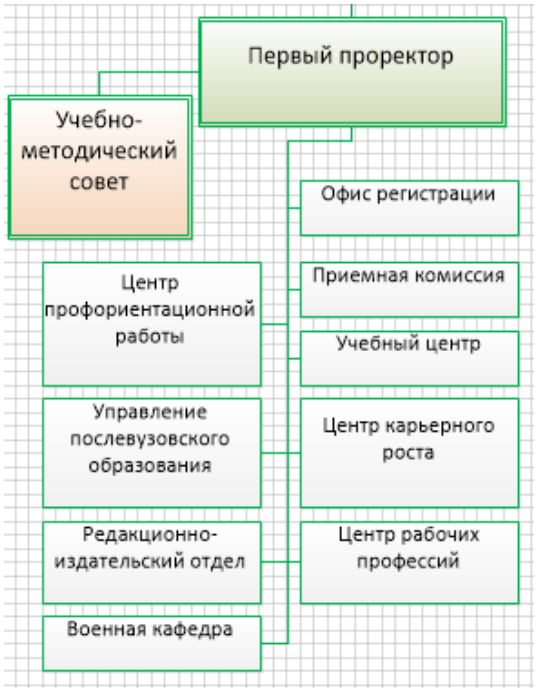


Рисунок 67 – Часть диаграммы «Первый проректор»

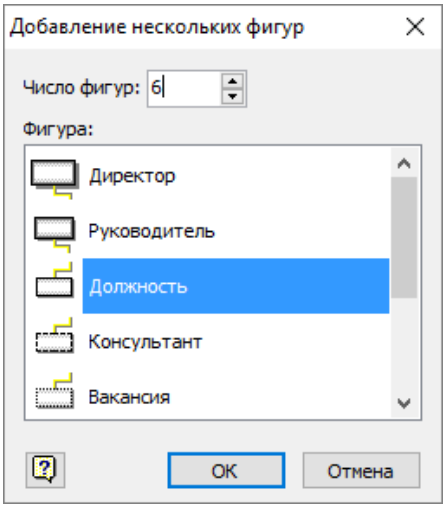


Рисунок 68 – Добавление нескольких фигур

10.Повторите предыдущий шаг и создайте полную организационную диаграмму (рис. 10-13).

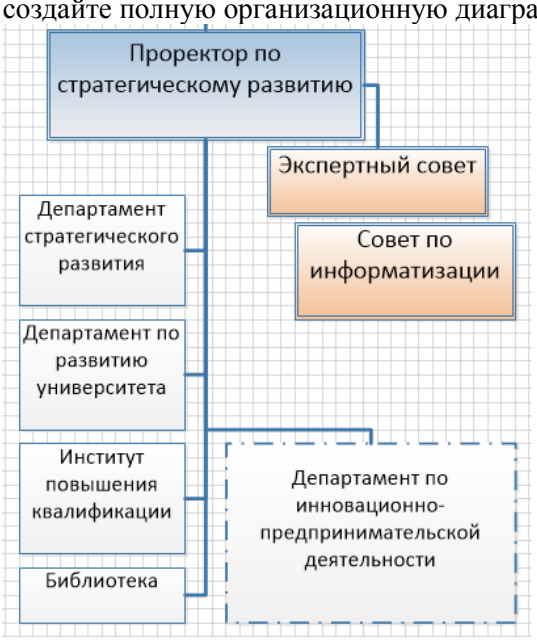


Рисунок 69 – Часть диаграммы «Проректор по стратегическому развитию»

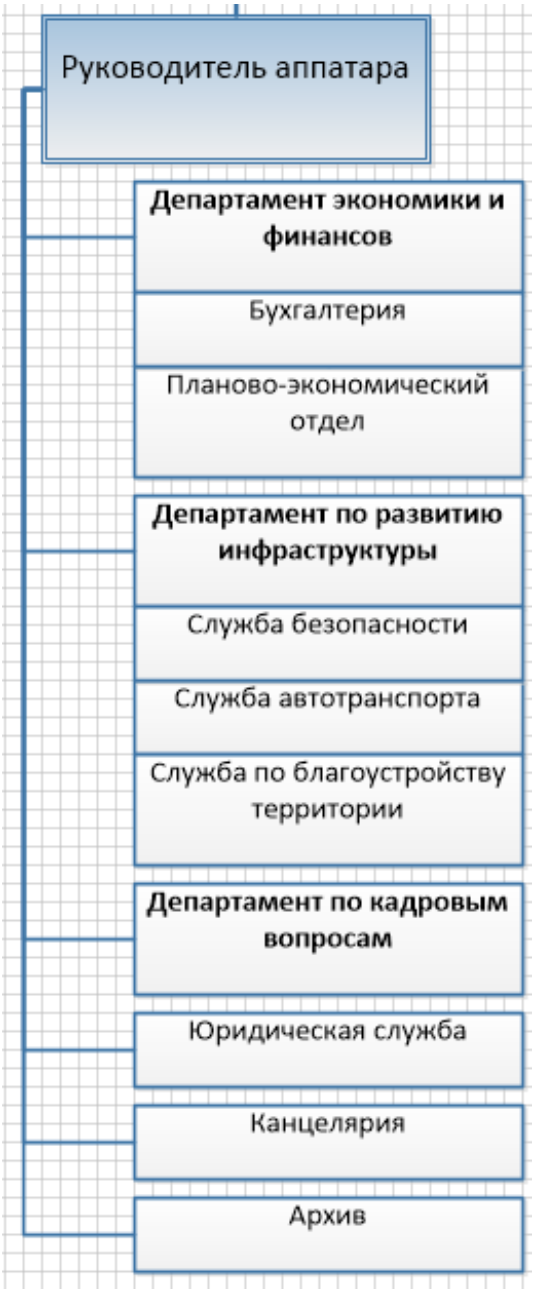


Рисунок 70 – Часть диаграммы «Руководитель аппарата»

Рисунок 71 – Часть диаграммы «Проректор по научной работе»

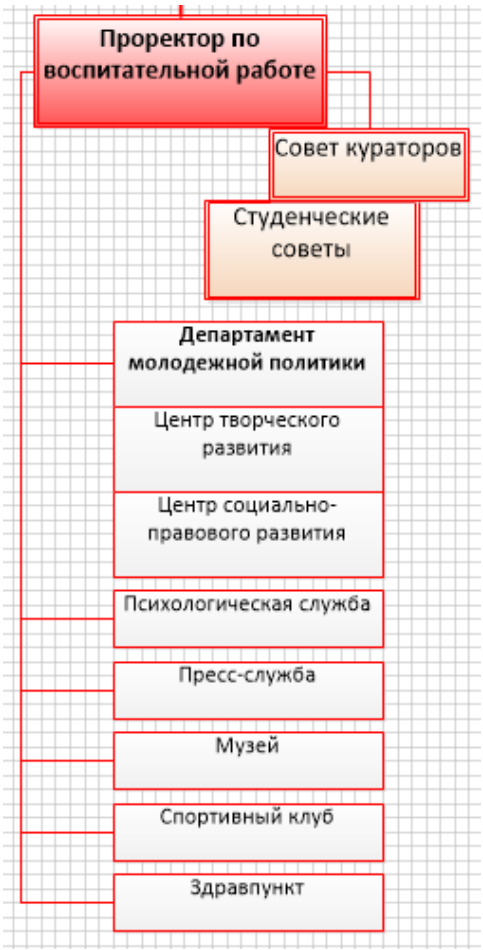


Рисунок 72 – Часть диаграммы «Проректор по ВР»

11.В результате выполнения всех описанных шагов организационная диаграмма ВУЗа должна принять вид, представленный на рисунке 14

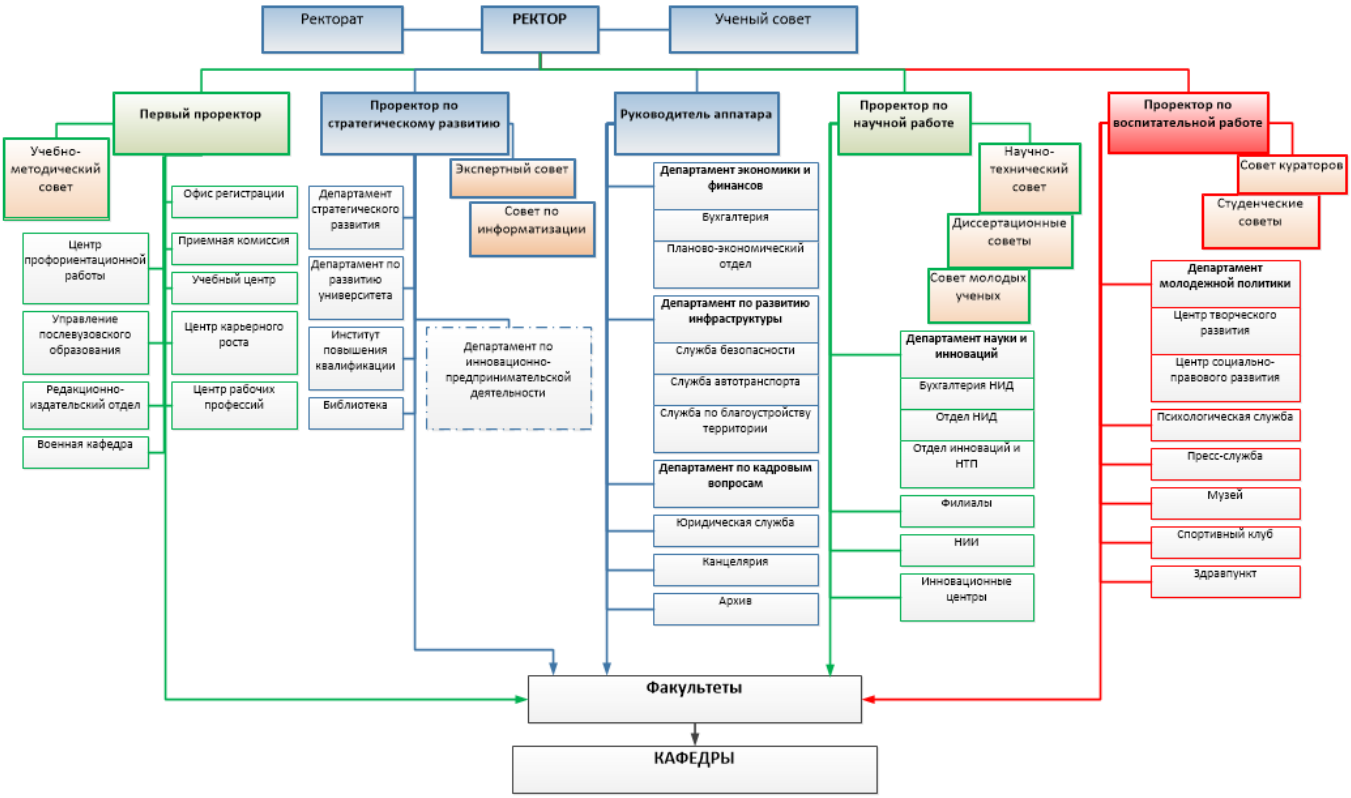


Рисунок 73 – Организационная диаграмма ВУЗа

4.2 Создание организационной диаграммы с помощью мастера диаграмм на основе данных

Построение организационной диаграммы с помощью мастера рассмотрим также на примере ВУЗа, однако немного упростим структуру.

Для построения организационной диаграммы с помощью мастера диаграмм необходимо проделать следующие действия:

1. Запустить *MS Visio*.
2. В разделе *Выберите шаблон* выбрать категорию *Бизнес*, а затем *Мастер организационных диаграмм* (рис. 15) и нажать кнопку *Создать*.

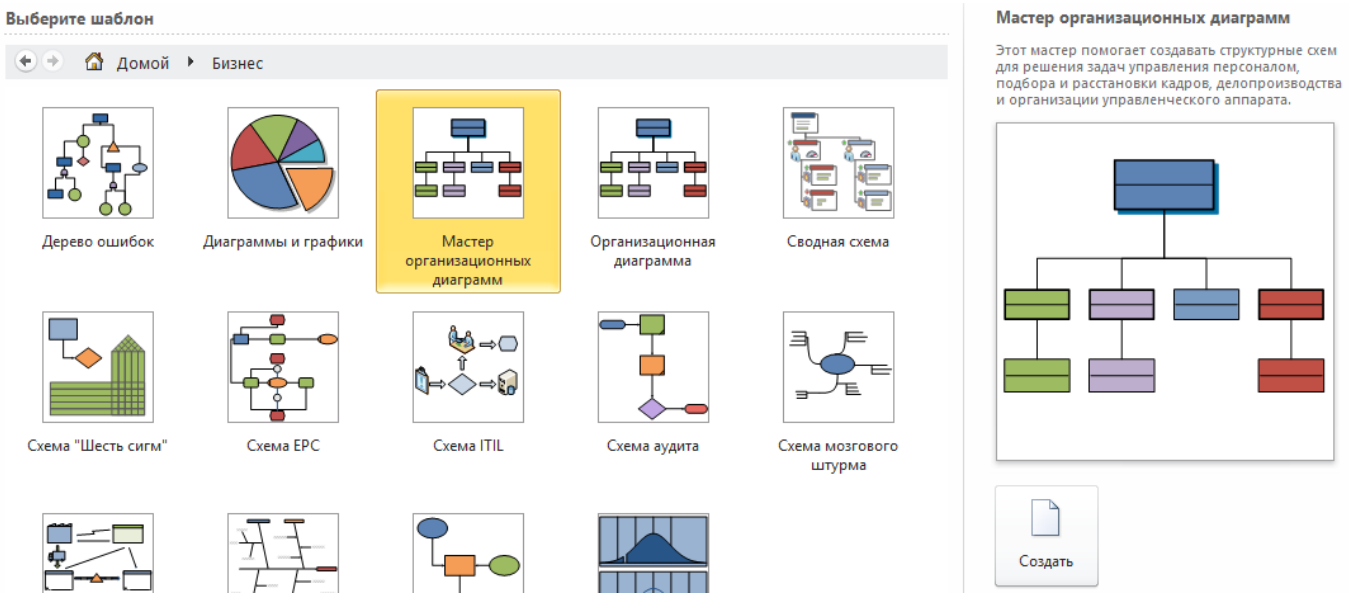


Рисунок 74 – Выбор шаблона «Мастер организационных диаграмм»

3. На первой странице мастера организационных диаграмм выберем переключатель *По данным, введенным с помощью мастера* (рис. 16). Обратите внимание – в описании этого переключателя подтверждается, что будет создан новый источник данных.

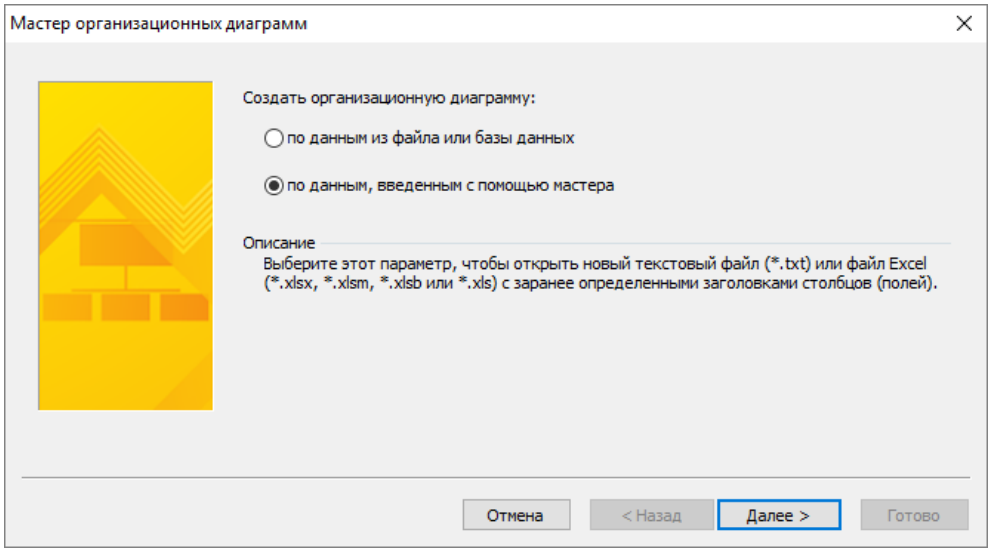


Рисунок 75 – Мастер организационных диаграмм. Шаг 1

4. Далее необходимо нажать кнопку *Далее* и выбрать тип файла.

На странице выбора типа файла выберем переключатель *Excel* и щелкнем на кнопке *Обзор*. В открывшемся диалоговом окне выберем папку, в которую нужно сохранить файл, в поле *Имя файла* введем *Данные оргдиаграммы* и щелкнем на кнопке Сохранить. Имя файла отображается в поле *Имя нового файла*. (рис. 17).

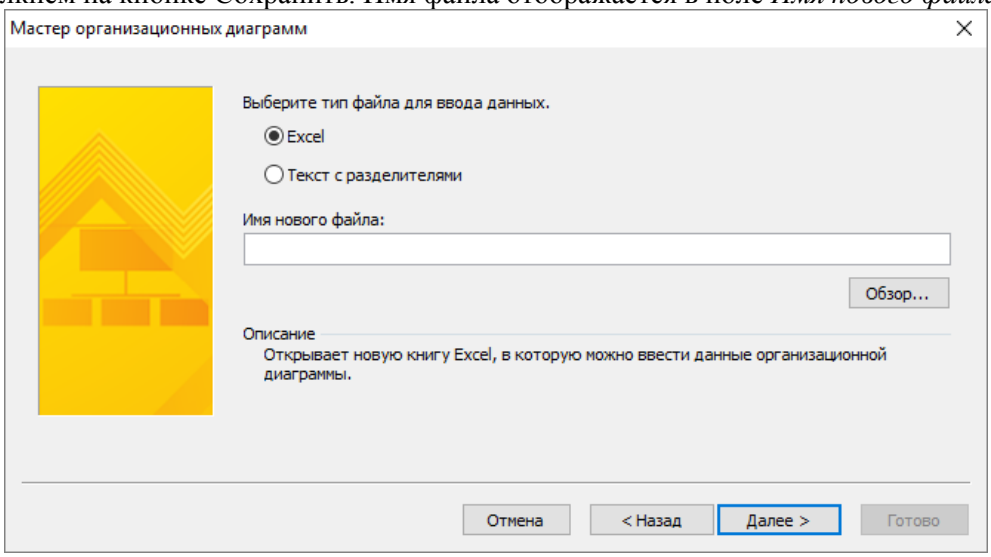


Рисунок 76 – Мастер организационных диаграмм. Шаг 2

5. Щелкнем на кнопке *Далее*. MS Visio показывает на необходимость ввести свои данные поверх ранее введенных (рис. 18).

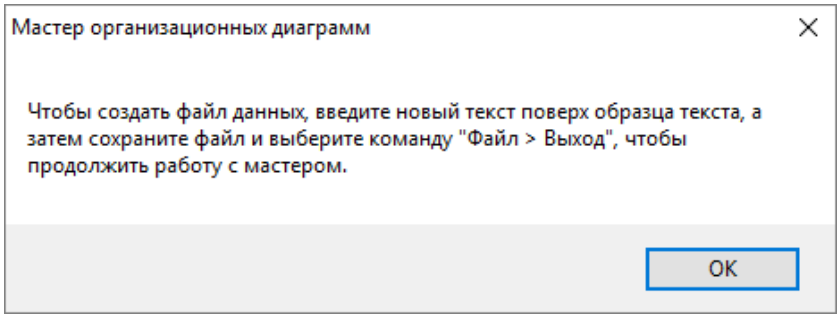


Рисунок 77 – Мастер организационных диаграмм. Шаг 3

6. Щелкнем на кнопке *Ок*. В Excel отображается форматированная книга. ***На рисунке 19 показана книга Excel для ввода данных диаграммы, при этом заголовок каждого столбца включает примечание с инструкциями по вводу данных в этом столбце.***

	A	B	C	D	E
1	Имя	Руководитель	Должность	ОТДЕЛ	ТЕЛЕФОН
2					
3	Сергей Белов		Ге	Директор	x5555
4	Ольга Петрова	Сергей Белов	Руководитель разработки	Разработка продукта	x6666
5	Игорь Сергеев	Ольга Петрова	Разработчик программного обеспечения	Разработка продукта	x6667
6					

Введите фамилию лица, которому подчиняется данный сотрудник, как оно показано в поле

Рисунок 78 – Форматированная книга Excel по умолчанию

7. Далее вносим свои данные в соответствии с рисунком 20.

	A	B	C	D	E
1	Имя	Руководитель	ДОЛЖНОСТЬ	ОТДЕЛ	ТЕЛЕФОН
2					
3	Иванов Иван		Ректор	Ректорат	
4	Петров Петр	Иванов Иван	Ректорат	Ректорат	
5	Смирнов Андрей	Иванов Иван	Ученый совет	Ректорат	
6	Сидоров Михаил	Иванов Иван	Первый проректор	Ректорат	
7	Кожемяко Галина	Иванов Иван	Проректор по стратегическому развитию	Ректорат	
8	Куликова Наталья	Иванов Иван	Руководитель аппарата	Ректорат	
9	Лымарь Александра	Иванов Иван	Проректор по научной работе	Ректорат	
10	Голубенко Артем	Иванов Иван	Проректор по воспитательной работе	Ректорат	
11	Евидченко Денис	Сидоров Михаил	Начальник приемной комиссии	Приемная комиссия	
12	Бершатская Светлана	Сидоров Михаил	Начальник военной кафедры	Военная кафедра	
13	Маньшина Кристина	Кожемяко Галина	Начальник департамента развития	Департамент стратегического развития	
14	Шмелева Оксана	Кожемяко Галина	Библиотекарь	Библиотека	
15	Игнатенко Денис	Кожемяко Галина	Начальник департамента экономики	Департамент экономики и финансов	
16	Ненашев Виктор	Куликова Наталья	Начальник службы безопасности	Департамент по развитию инфраструктуры	
17	Ткачев Анатолий	Куликова Наталья	Начальник отдела кадров	Департамент по кадровым вопросам	
18	Сидоренко Александр	Куликова Наталья	Архивариус	Архив	
19	Акинин Павел	Лымарь Александра	Начальник департамента науки	Департамент науки и инноваций	
20	Смирнова Оксана	Голубенко Артем	Начальник департамента молодежной политики	Департамент молодежной политики	
21	Ермаков Дмитрий	Голубенко Артем	Начальник психологической службы	Психологическая служба	
22	Павлова Мария	Голубенко Артем	Врач	Здравпункт	

Рисунок 79 – Данный организационной диаграммы

8. Закрываем Excel. После закрытия Excel происходит возврат в программу MS Visio, где открыта страница импорта фотографий мастера (рис. 21).

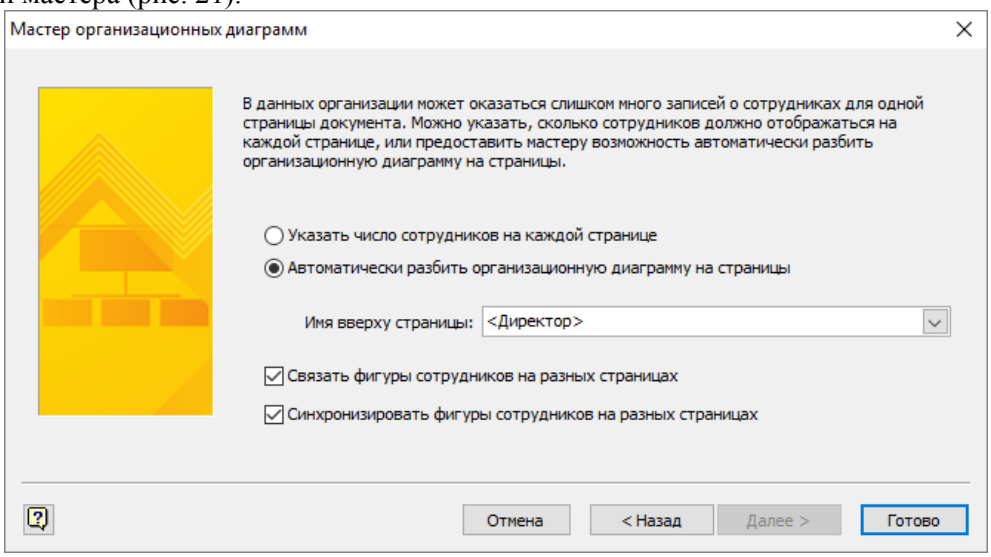


Рисунок 80 – Мастер организационных диаграмм. Шаг 4

9. Щелкнем на кнопке *Готово*, чтобы отобразить организационную диаграмму (рисунок 22).

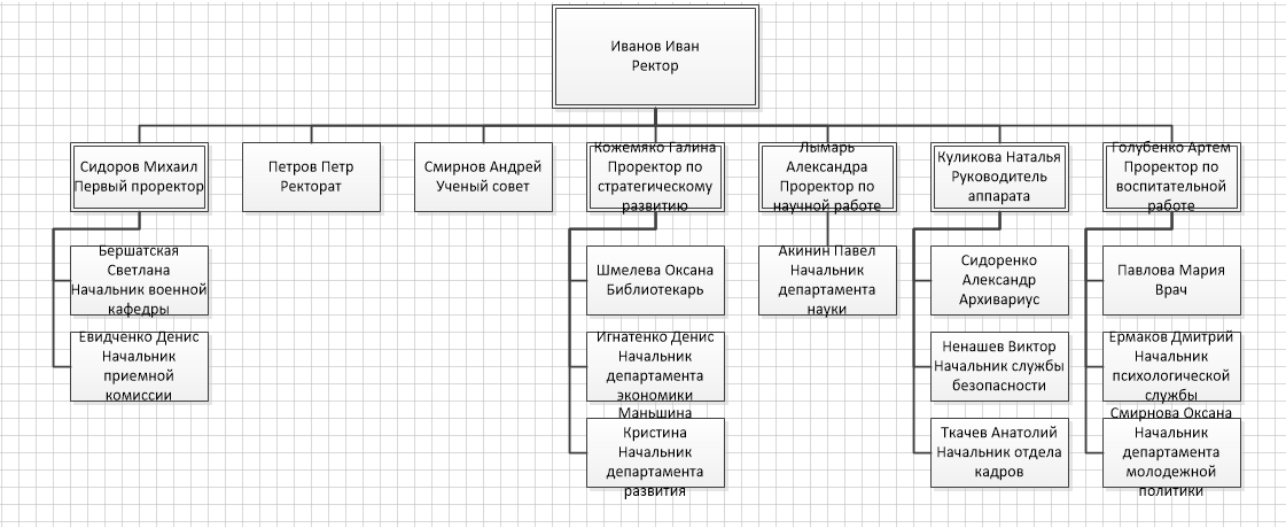


Рисунок 81 – Сформированная оргдиаграмма

10.На последнем шаге необходимо произвести форматирование диаграммы. В результате организационная диаграмма должна принять вид, показанный на рисунке 23.

Рисунок 82 – Итоговый вид диаграммы

5. Задание

Построить организационную диаграмму в соответствии в вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения организационной диаграммы, а также скриншоты результатов согласно заданию.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Что такое организационная диаграмма?
2. Способы построения оргдиаграмм в MS Visio?
3. Каковы принципы создания организационных диаграмм в MS Visio?
4. Какие существуют типы организационных структур? Перечислите их преимущества и недостатки.

Практическая работа №7

Построение диаграмм прецедентов на языке UML с помощью MS Visio

1. Цель работы

Целью работы является изучение основ создания диаграмм прецедентов (вариантов использования) на языке UML.

2. Задачи

Основными задачами практической работы являются:

- ознакомиться с теоретическими вопросами построения диаграмм прецедентов с помощью MS Visio;
- получить навыки создания диаграмм прецедентов.

3. Краткие теоретические сведения

3.1. Общие сведения о языке UML

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем.

Язык UML одновременно является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения.

В языке UML используется ***четыре основных вида графических конструкций***:

- **Значки или пиктограммы**. Значок представляет собой графическую фигуру фиксированного размера и формы. Примерами значков могут служить окончания связей элементов диаграмм или некоторые другие дополнительные обозначения.
- **Графические символы на плоскости**. Такие двумерные символы изображаются с помощью некоторых геометрических фигур и могут иметь различную высоту и ширину с целью размещения внутри этих фигур других конструкций языка UML.

Наиболее часто внутри таких символов помещаются строки текста, которые уточняют семантику или фиксируют отдельные свойства соответствующих элементов языка UML. Информация, содержащаяся внутри фигур, имеет важное значение для конкретной модели проектируемой системы, поскольку регламентирует реализацию соответствующих элементов в программном коде.

- **Пути,** которые представляют собой последовательности из отрезков линий, соединяющих отдельные графические символы. При этом концевые точки отрезков линий должны обязательно соприкасаться с геометрическими фигурами, служащими для обозначения вершин диаграмм, как принято в теории графов. С концептуальной точки зрения путям в языке UML придается особое значение, поскольку они являются простыми топологическими сущностями.
- **Строки текста.** Служат для представления различных видов информации в некоторой грамматической форме. Предполагается, что каждое использование строки текста должно соответствовать синтаксису в нотации языка UML, посредством которого может быть реализован грамматический разбор этой строки.

При графическом изображении диаграмм следует придерживаться следующих ***основных рекомендаций***:

1. Каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области.
2. Все сущности на диаграмме модели должны быть одного концептуального уровня. Здесь имеется в виду согласованность не только имен одинаковых элементов, но и возможность вложения отдельных диаграмм друг в друга для достижения полноты представлений.
3. Вся информация о сущностях должна быть явно представлена на диаграммах. Речь идет о том, что, хотя в языке UML при отсутствии некоторых символов на диаграмме могут быть использованы их значения по умолчанию (например, в случае неявного указания видимости атрибутов и операций классов), необходимо стремиться к явному указанию свойств всех элементов диаграмм.
4. Диаграммы не должны содержать противоречивой информации. Противоречивость модели может служить причиной серьезнейших проблем при ее реализации и последующем использовании на практике. Например, наличие замкнутых путей при изображении отношений агрегирования или композиции приводит к ошибкам в программном коде, который будет реализовывать соответствующие классы. Наличие элементов с одинаковыми именами и различными атрибутами свойств в одном пространстве имен также приводит к неоднозначной интерпретации и может служить источником проблем.
5. Диаграммы не следует перегружать текстовой информацией. Принято считать, что визуализация модели является наиболее эффективной, если она содержит минимум пояснительного текста.
6. Каждая диаграмма должна быть самодостаточной для правильной интерпретации всех ее элементов и понимания семантики всех используемых графических символов.

7. Количество типов диаграмм для конкретной модели приложения не является строго фиксированным.

3.2. Диаграмма вариантов использования (usecase diagram)

Разработка диаграммы вариантов использования преследует цели:

1. Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
2. Сформулировать общие требования к функциональному поведению проектируемой системы.
3. Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
4. Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования.

При этом **актером (actor)** или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик.

В свою очередь, **вариант использования (usecase)** служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

В самом общем случае, диаграмма вариантов использования представляет собой граф специального вида, который является графической нотацией для представления конкретных вариантов использования, актеров, возможно некоторых интерфейсов, и отношений между этими элементами.

Вариант использования (use case) **–** конструкция или стандартный элемент языка UML, который применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером.

Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов.

Такой пояснительный текст получил название ***примечания или сценария*****.**

Отдельный вариант использования обозначается на диаграмме эллипсом (рис. 1), внутри которого содержится его краткое название или имя в форме глагола с пояснительными словами.

Рисунок 83 – Графическое обозначение варианта использования

Актер (actor) представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования. Стандартным графическим обозначением актера на диаграммах является фигурка «человечка» (рис. 2), под которой записывается конкретное имя актера.

Рисунок 84 – Графическое обозначение актера

Интерфейс (interface) служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры. В языке UML интерфейс является классификатором и характеризует только ограниченную часть поведения моделируемой сущности. Применительно к диаграммам вариантов использования, интерфейсы определяют совокупность операций, которые обеспечивают необходимый набор сервисов или функциональности

для актеров. Интерфейсы не могут содержать ни атрибутов, ни состояний, ни направленных ассоциаций. Они содержат только операции без указания особенностей их реализации.

На диаграмме вариантов использования интерфейс изображается в виде маленького круга, рядом с которым записывается его имя (рис. 3).

Рисунок 85 – Графическое обозначение интерфейса

В качестве имени может быть существительное, которое характеризует соответствующую информацию или сервис (например, «датчик», «сирена», «видеокамера»), но чаще строка текста (например, «запрос к базе данных», «форма ввода», «устройство подачи звукового сигнала»).

Если имя записывается на английском, то оно должно начинаться с заглавной буквы I, например, ISecurelnformation, ISensor.

Графический символ отдельного интерфейса может соединяться на диаграмме сплошной линией с тем вариантом использования, который его поддерживает. Сплошная линия в этом случае указывает на тот факт, что связанный с интерфейсом вариант использования должен реализовывать все операции, необходимые для данного интерфейса, а возможно и больше (рис. 4а).

Кроме этого, интерфейсы могут соединяться с вариантами использования пунктирной линией со стрелкой (рис. 4б), означающей, что вариант использования предназначен для спецификации только того сервиса, который необходим для реализации данного интерфейса.

Рисунок 86 – Графическое обозначение взаимосвязей интерфейсов и вариантов использования

Примечания (notes) в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры сущностей) и помеченные значения.

Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Графически примечания обозначаются прямоугольником с «загнутым» верхним правым уголком (рис. 5). Внутри прямоугольника содержится текст примечания. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия. Если примечание относится к нескольким элементам, то от него проводятся, соответственно, несколько линий. Разумеется, примечания могут присутствовать не только на диаграмме вариантов использования, но и на других канонических диаграммах.

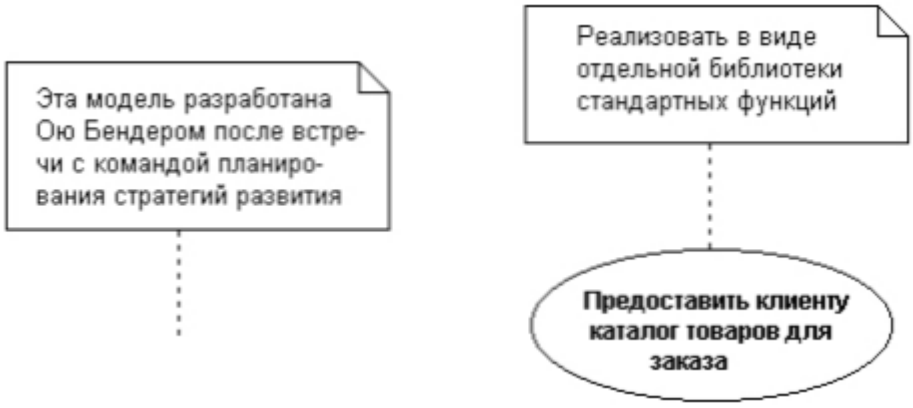


Рисунок 87 – Пример обозначения примечания

3.3. Отношения на диаграмме вариантов использования

Между компонентами диаграммы вариантов использования могут существовать различные отношения, которые описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов.

Один актер может взаимодействовать с несколькими вариантами использования. В этом случае этот актер обращается к нескольким сервисам данной системы. В свою очередь один вариант использования может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис. Следует заметить, что два варианта использования, определенные для одной и той же сущности, не могут взаимодействовать друг с другом, поскольку каждый из них самостоятельно описывает законченный вариант использования этой сущности. Более того, варианты использования всегда предусматривают некоторые сигналы или сообщения, когда взаимодействуют с актерами за пределами системы. В то же время могут быть определены другие способы для взаимодействия с элементами внутри системы.

В языке UML имеется несколько стандартных ***видов отношений между актерами и вариантами использования:***

1. Отношение ассоциации (association relationship)

Отношение ассоциации является одним из фундаментальных понятий в языке UML и в той или иной степени используется при построении всех графических моделей систем в форме канонических диаграмм.

Применительно к диаграммам вариантов использования оно служит для обозначения специфической роли актера в отдельном варианте использования. Другими словами, ассоциация специфицирует семантические особенности взаимодействия актеров и вариантов использования в графической модели системы. Таким образом, это отношение устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования. На диаграмме вариантов использования, так же, как и на других диаграммах, отношение ассоциации обозначается сплошной линией между актером и вариантом использования. Эта линия может иметь дополнительные условные обозначения, такие, например, как имя и кратность (рис. 6).

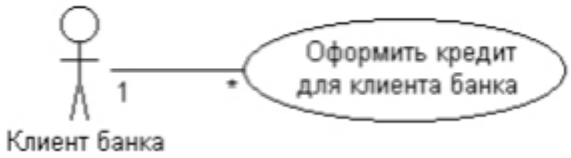


Рисунок 88 – Графическое обозначение отношения ассоциации

2. Отношение расширения (extend relationship)

Отношение расширения определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров.

Так, если имеет место отношение расширения от варианта использования А к варианту использования В, то это означает, что свойства экземпляра варианта использования В могут быть дополнены благодаря наличию свойств у расширенного варианта использования А.

Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом «extend» («расширяет»), как показано на рис. 7

Рисунок 89 – Графическое обозначение отношения расширения

Отношение расширения отмечает тот факт, что один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования.

Один из вариантов использования может быть расширением для нескольких базовых вариантов, а также иметь в качестве собственных расширений несколько других вариантов. Базовый вариант использования может дополнительно никак не зависеть от своих расширений.

3. Отношение обобщения (generalization relationship)

Отношение обобщения служит для указания того факта, что некоторый вариант использования А может быть обобщен до варианта использования В.

В этом случае вариант А будет являться специализацией варианта В. При этом В называется предком или родителем по отношению А, а вариант А – потомком по отношению к варианту использования В. Следует подчеркнуть, что потомок наследует все свойства и поведение своего родителя, а также может быть дополнен новыми свойствами и особенностями поведения.

Графически данное отношение обозначается сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительский вариант использования (рис. 8). Эта линия со стрелкой имеет специальное название – стрелка «обобщение».

Рисунок 90 – Графическое обозначение отношения обобщения

Отношение обобщения между вариантами использования применяется в том случае, когда необходимо отметить, что дочерние варианты использования обладают всеми атрибутами и особенностями поведения родительских вариантов. При этом дочерние варианты использования участвуют во всех отношениях родительских вариантов. В свою очередь, дочерние варианты могут наделяться новыми свойствами поведения, которые отсутствуют у родительских вариантов использования, а также уточнять или модифицировать наследуемые от них свойства поведения.

Между отдельными актерами также может существовать отношение обобщения. Данное отношение является направленным и указывает на факт специализации одних актеров относительно других. Например, отношение обобщения от актера А к актеру В отмечает тот факт, что каждый экземпляр актера А является одновременно экземпляром актера В и обладает всеми его свойствами. В этом случае актер В является родителем по отношению к актеру А, а актер А, соответственно, потомком актера В. При этом актер А обладает способностью играть такое же множество ролей, что и актер В.

Графически данное отношение также обозначается стрелкой обобщения, т. е. сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительского актера (рис. 9).

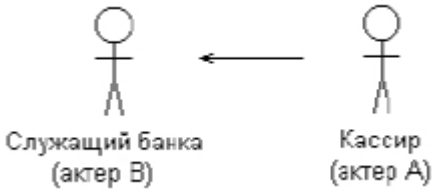


Рисунок 91 – Графическое обозначение отношения обобщения между актерами

4. Отношение включения (include relationship)

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента последовательность поведения другого варианта использования. Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в отношении включения.

Отношение включения, направленное от варианта использования А к варианту использования В, указывает, что каждый экземпляр варианта А включает в себя функциональные свойства, заданные для варианта В. Эти свойства специализируют поведение соответствующего варианта А на данной диаграмме. Графически данное отношение обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом «include» («включает»), как показано на рис. 10

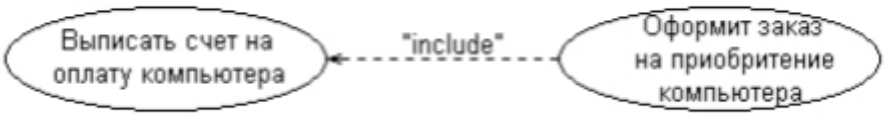


Рисунок 92 – Графическое обозначение отношения включения

5. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

5. Запустите MS Visio.
6. На экране выбора шаблона выберите категорию *Программы и БД* и в ней элемент *Схема модели UML*. Нажмите кнопку *Создать* в правой части экрана.
7. Окно программы примет вид, подобный рис. 11.

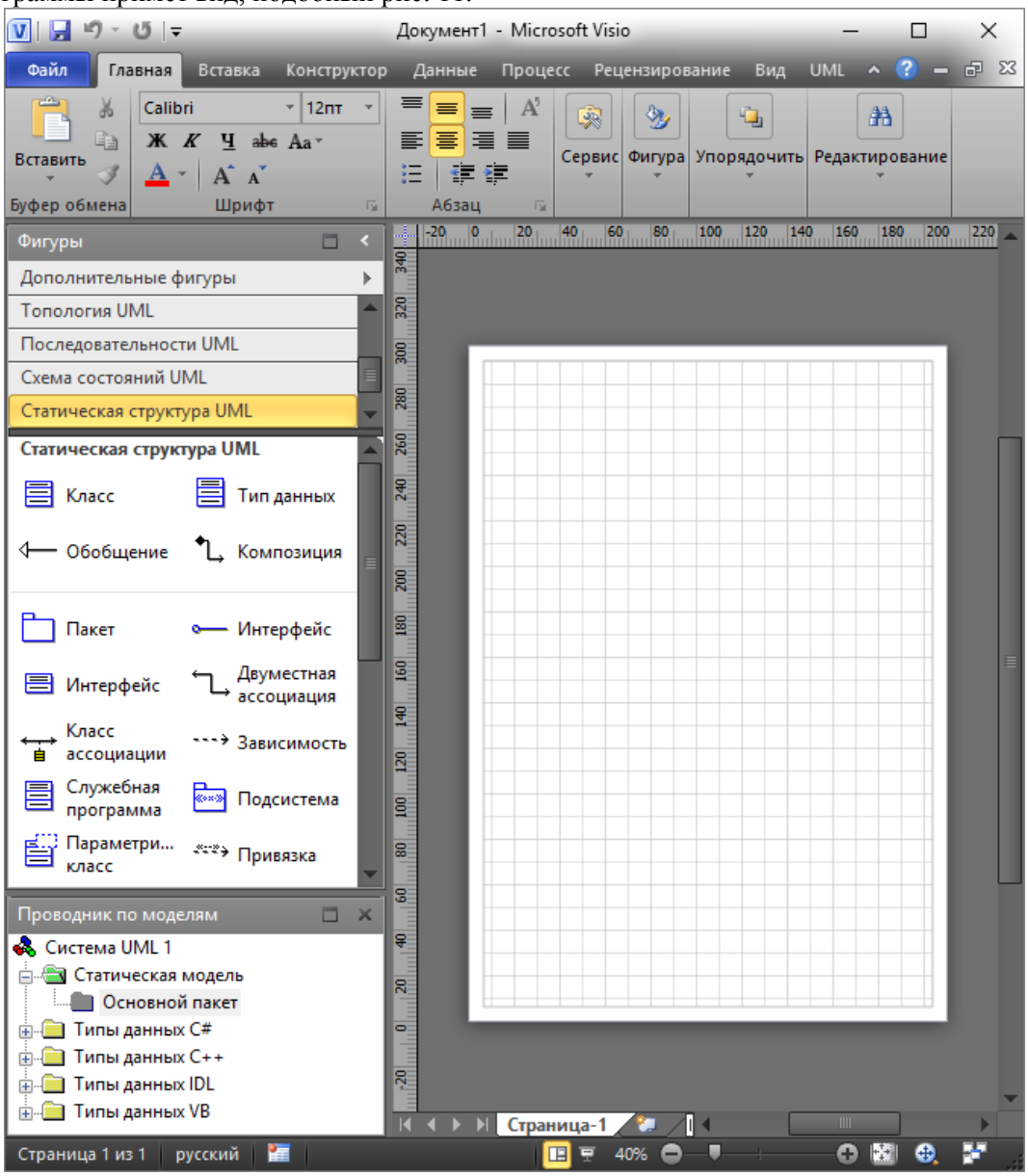


Рисунок 93 – Схема модели UML в MS Visio

8. Далее необходимо открыть все фигуры, необходимые для построения UML-диаграмм. Для этого в левой части экрана необходимо нажать кнопку *Дополнительные фигуры*. В открывшемся вспомогательном меню выбрать *Программы и БД -> Программное обеспечение* и выбрать все доступные фигуры для построения UML (рис. 12).

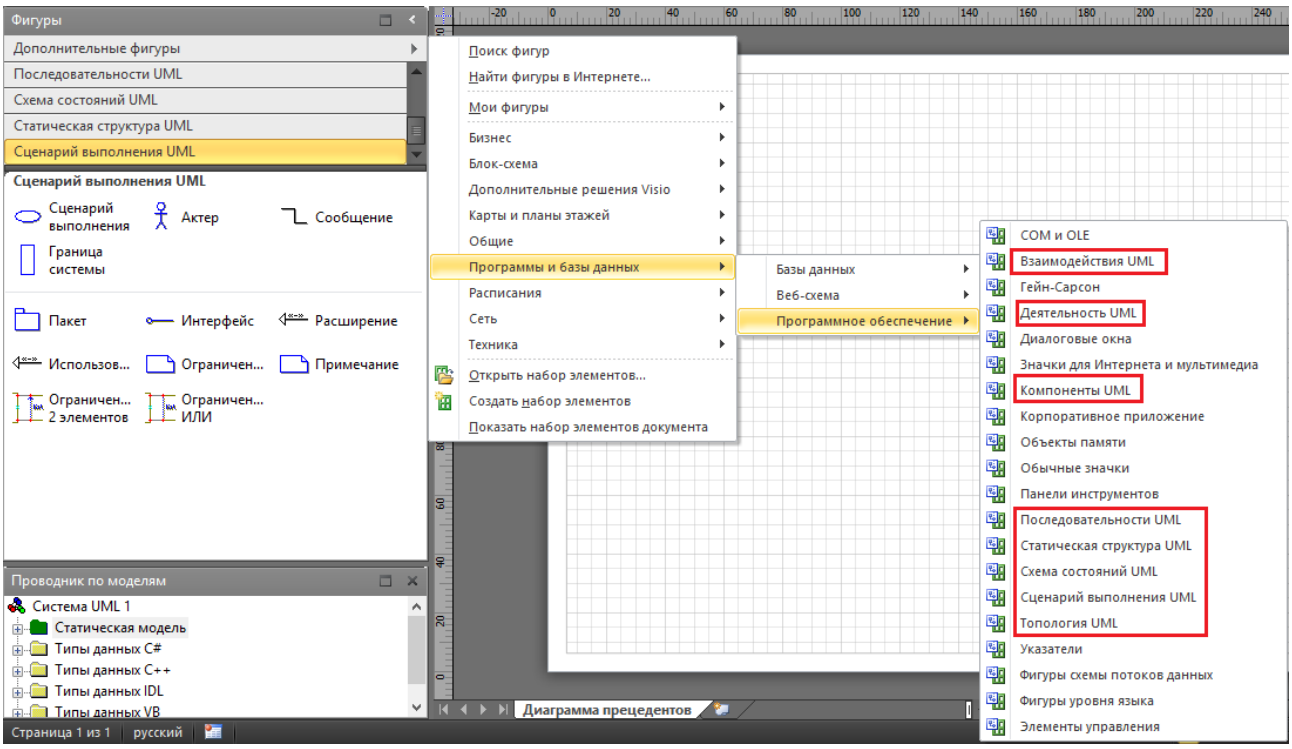


Рисунок 94 – Добавление фигур UML

9. После этого необходимо провести следующие этапы моделирования.

9.1. Выбор актеров.

В качестве актеров данной системы могут выступать два субъекта, один из которых является продавцом, а другой – покупателем. Каждый из этих актеров взаимодействует с рассматриваемой системой продажи товаров по каталогу и является ее пользователем, т. е. они оба обращаются к соответствующему сервису «Оформить заказ на покупку товара». Как следует из существа выдвигаемых к системе требований, этот сервис выступает в качестве варианта использования разрабатываемой диаграммы, первоначальная структура которой может включать в себя только двух указанных актеров и единственный вариант использования (рис. 14).

– В группе фигур *Сценарий выполнения UML* выбрать блок *Граница системы* и добавить его на лист.

– Внутрь границы системы добавить блок *Сценарий выполнения* и добавить к нему название, дважды щелкнув внутри блока.

– Добавить два блока *Актер* – покупатель и продавец.

– С помощью блока *Сообщение* установите связь актеров и варианта использования. Двойным щелчком правой кнопки мыши по блоку Сообщение откройте окно *Свойств ассоциации UML,* проведите настройки в соответствии с рисунком 13.

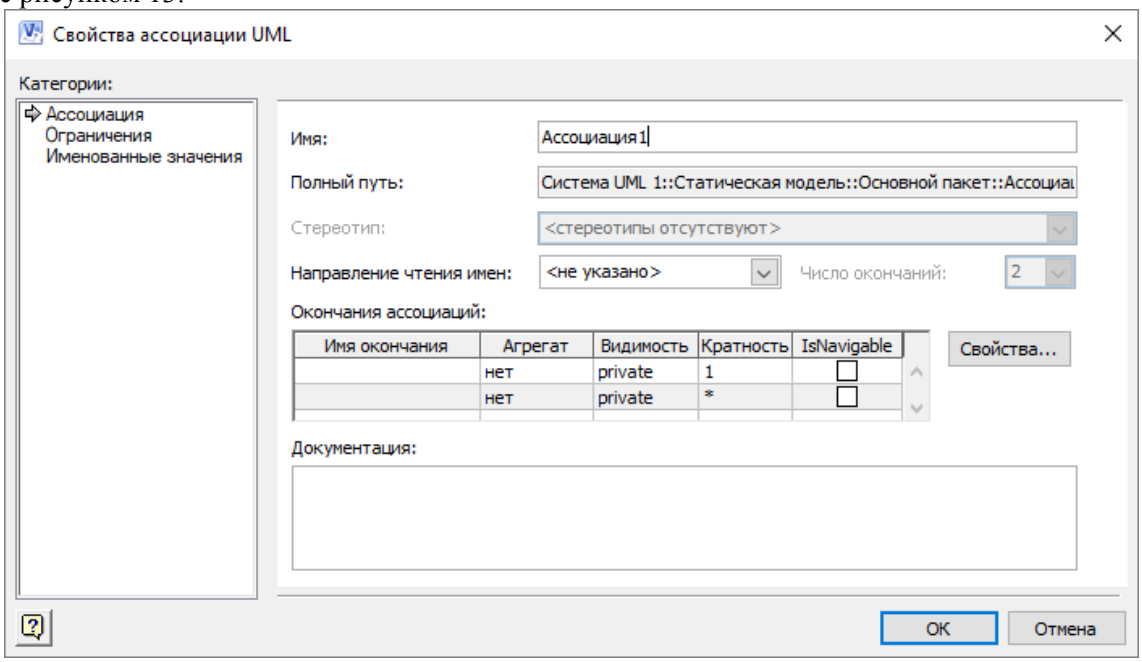


Рисунок 95 – Свойства ассоциации UML

Рисунок 96 – Исходная диаграмма вариантов использования системы по продаже товаров

9.2. Выделение дополнительных вариантов использования.

Детализировать вариант использования «Оформить заказ на продажу товара» можно выделив следующие дополнительные варианты использования:

- обеспечить покупателя информацией – является отношением включения;
- согласовать условия оплаты – является отношением включения;
- заказать товар со склада – является отношением включения;
- запросить каталог товаров – является отношением расширения.

Так как в MS Visio отсутствует отношение включения, его необходимо добавить самостоятельно. Для этого перейти на вкладку *UML* -> в группе *Модель* выбрать пункт *Стереотипы.* В открывшемся окне нажать кнопку *Создать* и настроить стереотип в соответствии с рисунком 15.

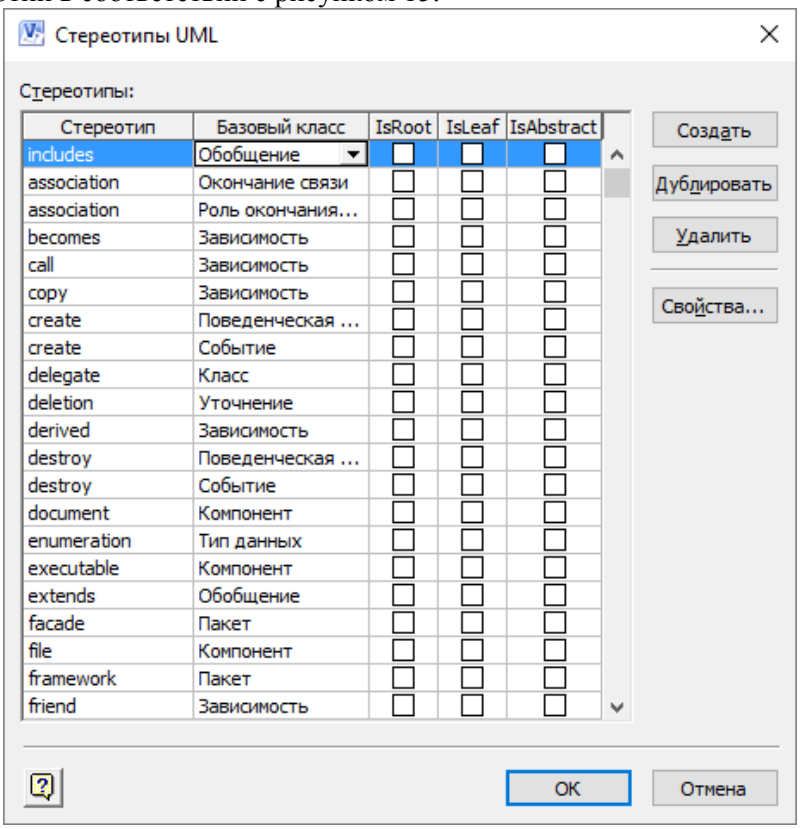


Рисунок 97 – Создание стереотипа

Далее на новом листе необходимо добавить границу системы и все варианты использования. После чего соединить варианты использования с помощью блока *Расширение.*

Для того, чтобы изменить тип отношения дважды щелкните по стрелке и окне свойств задайте необходимые параметры.

Дополненная диаграмма вариантов использования примет вид, показанный на рисунке 16.

Рисунок 98 – Дополненная диаграмма вариантов использования

9.3. Написание описательной спецификации для каждого варианта использования.

Спецификация для варианта использования «Оформить заказ на покупку компьютера» приведена в таблице 1.

Таблица 10 – Спецификация варианта использования

Раздел	**Описание**
Краткое описание	Покупатель желает оформить заказ на покупку компьютера, который он выбрал в каталоге товаров. При условии, что клиент зарегистрирован и выбранный компьютер есть в наличии оформляется заказ. Если клиент не зарегистрирован, то предлагается ему пройти регистрацию, и после этого заказать выбранный компьютер. Если компьютера нет в наличии, то предлагается заказать товар со склада в течении заданного срока поставки.
Субъекты	Продавец, Покупатель
Предусловия	В каталоге товаров имеются компьютеры, которые можно заказать. У покупателей есть доступ к системе для регистрации. Продавцы умеют пользоваться рассматриваемой системой продажи. У покупателя есть бонусы.
Основной поток	Зарегистрированный покупатель имеет возможность заказать любой компьютер из каталога товаров. В случае наличия выбранного компьютера оформляется заказ с присвоением ему уникального номера. После этого покупателю предлагается выбрать способ оплаты и способ получения компьютера. В случае отсутствия компьютера в наличии предлагается оформить заказ со склада и ожидания его поставки в рамках указанного срока или выбрать другой компьютер.
Альтернативный поток	Покупатель не зарегистрирован. В этом случае, прежде чем оформить заказ на компьютер, ему предлагается пройти регистрацию. Попытка заказать товар, который отсутствует на складе Начисление бонусов
Постусловия	Заказ оформлен и определен срок поставки компьютера и место его получения

На рисунках 17-19 приведены примеры диаграмм вариантов использования для различных систем.

Рисунок 99 – Пример 1

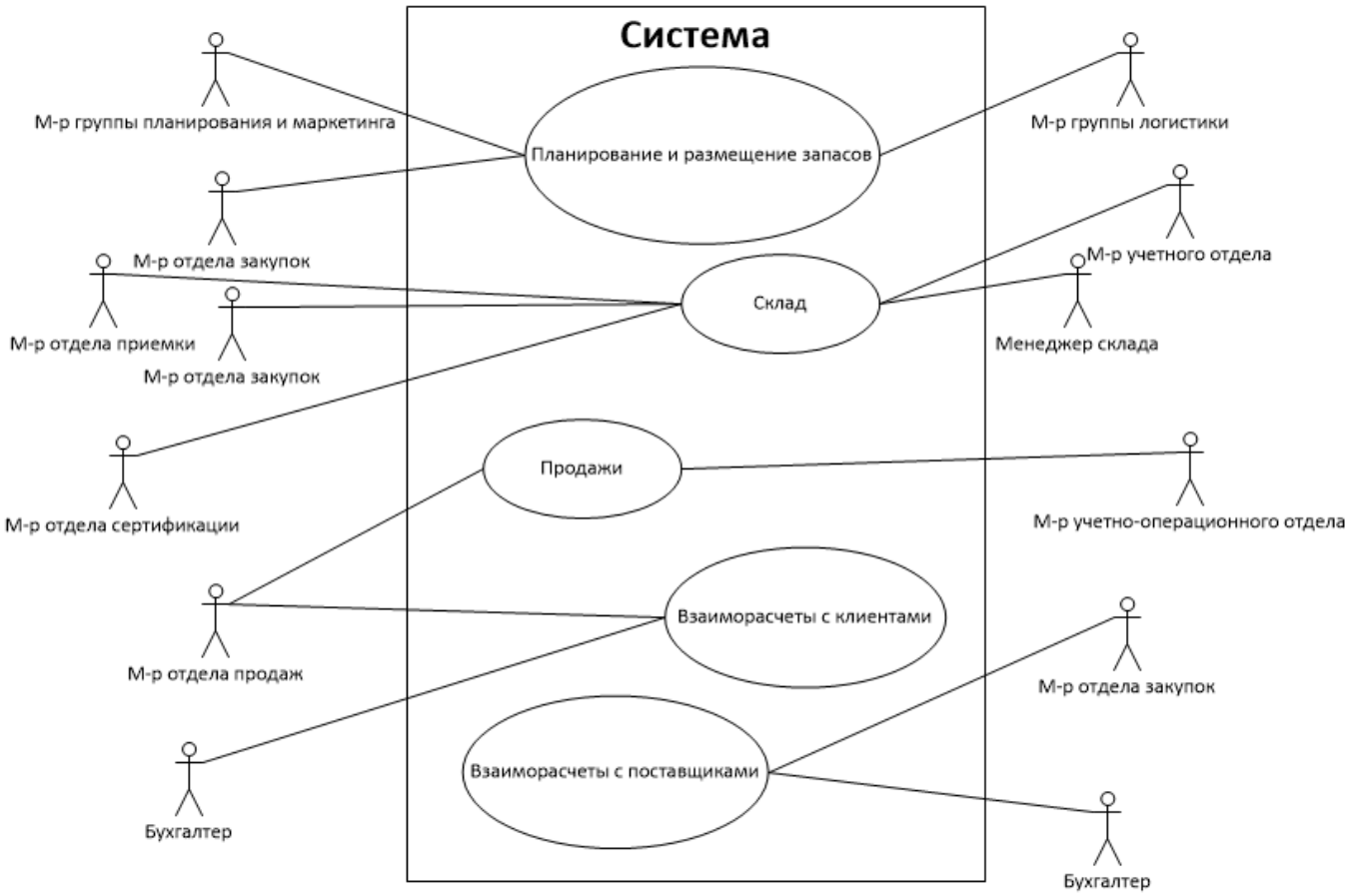


Рисунок 100 – Пример 2

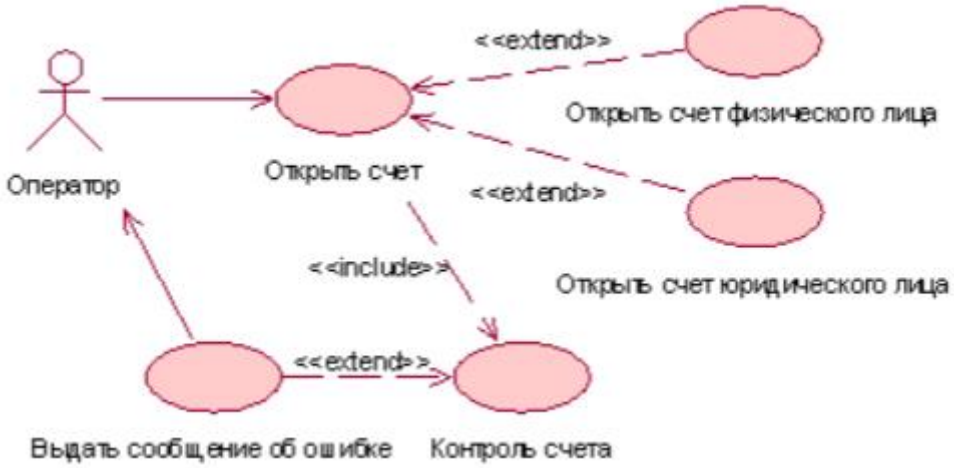


Рисунок 101 – Пример 3

6. Задание

Построить диаграмму прецедентов (вариантов использования) в соответствии с вариантом. Составить спецификцию.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения организационной диаграммы, а также скриншоты результатов согласно заданию.

7. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

8. Контрольные вопросы

1. Для чего используется язык UML?
2. Назначение диаграммы вариантов использования?
3. Что такое «актер»?
4. Что такое «вариант использования»?
5. Что такое «интерфейс»?
6. Что такое «примечание»?
7. Перечислить виды отношений между актерами и вариантами использования, охарактеризовать каждое из них?

Практическая работа №8

Построение диаграмм классов на языке UML с помощью MS Visio

1. Цель работы

Целью работы является изучение основ создания диаграмм классов на языке UML, получение навыков построения диаграмм классов, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

2. Задачи

Основными задачами практической работы являются:

- ознакомиться с теоретическими вопросами построения диаграмм классов;
- ознакомиться с теоретическими вопросами построения диаграмм классов с помощью MS Visio;
- получить навыки создания диаграмм классов.

3. Краткие теоретические сведения

Диаграмма классов является одной из канонических диаграмм UML, создаваемой для визуализации структурированной статической модели предметной области. Этот вид диаграмм представляет собой графическое изображение объектов – классов с присущими им атрибутами, операциями и различных отношений между классами.

3.1. Классы

Класс (class) служит для обозначения множества объектов, обладающих функциональным набором одинаково описывающих параметров (атрибутов), реализуемых операций и однотипными отношениями с объектами других классов.

На диаграмме класс изображается габаритной прямоугольной рамкой, которая дополнительно может быть разделена горизонтальными линиями на секции, каждая из которых предназначена для указания имени, атрибутов (свойств) и реализуемых операций объектов данного класса (рис. 1 а, б, в).

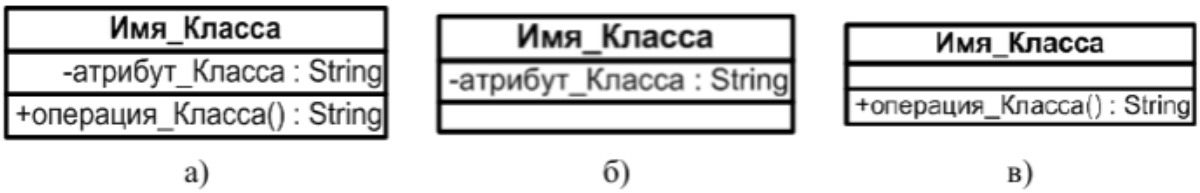


Рисунок 102 – а) Обозначение класса, б) Обозначение атрибутов, в) Обозначение операций

Имя класса является обязательным элементом в его обозначении и должно быть уникальным (хотя бы в пределах пакета), имеющее непосредственное отношение к контексту моделируемой предметной области.

В соответствии с принятым в языке UML общим соглашением в качестве имени класса используются существительные и прилагательные, каждое из которых начинается с заглавной буквы, записанные без пробелов. Например, в качестве имен классов могут быть использованы профессиональные термины: «Сотрудник», «Компания», «Руководитель», «Клиент», «Продавец», «Менеджер», «Офис», «Покупатель», «Датчик_Температуры» и др. Такое имена классов являются простыми.

Иногда возникает необходимость в явном указании пакета, к которому относится данный класс. С этой целью в условном обозначении перед именем класса указывается имя пакета и специальный символ разделитель – двойное двоеточие "::". Такое имя класса является квалифицированным. Текстовая строка имени класса в этом случае записывается в формате <Имя_пакета>::<Имя_класса>. Например, если определен пакет с именем «Банк», то имя класса «Счет» может быть записано так, как показано на рис. 2.

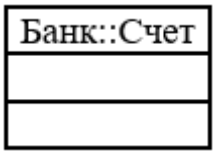


Рисунок 103 – Обозначение квалифицированного имени класса

3.2. Атрибуты классов

Содержательной характеристикой класса является атрибут, содержащий множество значений, которые могут принимать отдельные объекты этого класса. При этом, класс может иметь любое число атрибутов или не иметь ни одного. Так, например, атрибутами класса «Усилитель» являются частотный диапазон, выходная мощность, коэффициент нелинейных искажений, уровень шума и т. д.

Запись атрибута также представляет собой отдельную строку текста, содержащую обязательное имя, в котором обычно каждое слово пишется с заглавной буквы, за исключением первого, например, name (имя) или birth_Date (дата_Рождения).

Например, на рис. 3 указаны атрибуты класса Контейнер, в качестве которых выступают атрибут тип_Контейнера, атрибут регистрационный_Номер_Контейнера, регистрационный_Номер_После_Перерегистрации, рабочее_Состояние.

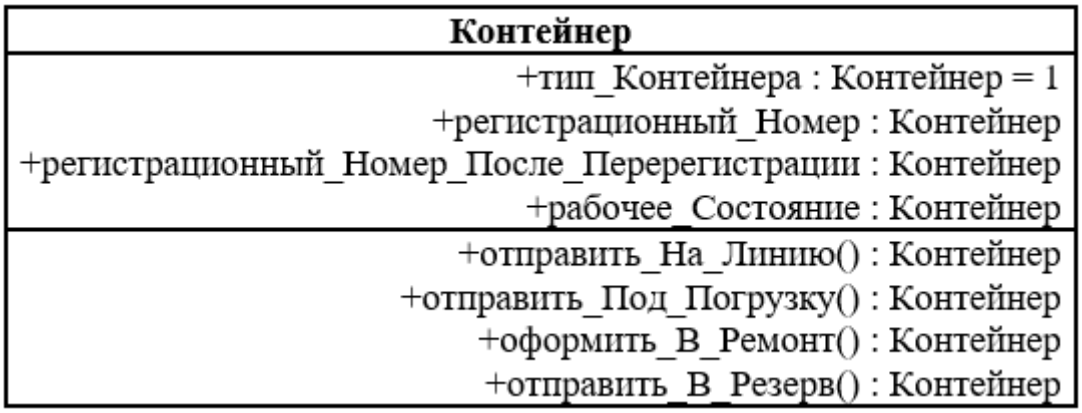


Рисунок 104 – Указание атрибутов класса Контейнер

Качественной характеристикой описания элементов класса является квантор видимости атрибута – потенциальная возможность других объектов модели оказывать влияние на отдельные аспекты поведения данного класса.

Эта характеристика может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов: символ "+" (public) обозначает атрибут с областью видимости типа общедоступный; атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма; например, для класса Class_1 указан атрибут общедоступного типа (рис. 4а); символ "#" (protected) обозначает атрибут с областью видимости типа защищенный; атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса; например, для класса Class_2 указан атрибут защищенного типа (рис. 4б); символ "-" (private) обозначает атрибут с областью видимости типа закрытый; атрибут с этой областью видимости недоступен или невиден для всех классов без исключения; например, для класса Class_3 указан атрибут защищенного типа (рис. 4в);

Квантор видимости при описании атрибутов может быть опущен, что будет означать тот факт, что видимость атрибута не указывается.

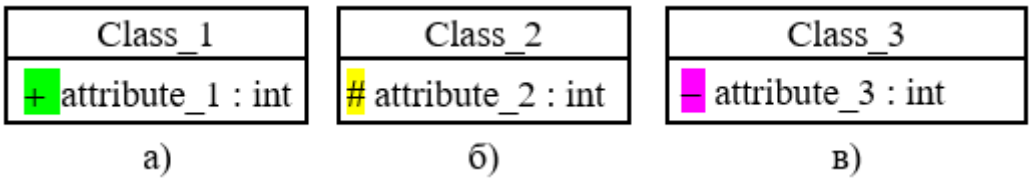


Рисунок 105 – Обозначение видимости атрибутов класса

3.3. Операции классов

Операция (operation) класса – это реализация услуги, которая может быть запрошена у любого объекта данного класса, чтобы вызвать определенное его поведение. Класс может иметь любое число операций либо не иметь ни одной. Так автомобиль может перемещаться по грунту, корабль – перемещаться по воде, компьютер – производить вычисления.

Представление полного синтаксиса записи операций класса также подчиняется определенным синтаксическим правилам: каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, обязательного имени операции, выражения типа возвращаемого операцией значения и, возможно, строки-свойства данной операции:

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

Квантор видимости, как и в случае атрибутов класса, может принимать одно из трех возможных значений и, соответственно, также отображается при помощи специального символа.

Для именования операции используются короткие глагольные конструкции, описывающие некоторое поведение класса, которому принадлежит операция. Обычно каждое слово в имени операции пишется с заглавной буквы, за исключением первого.

Например, запись +создать() – может обозначать абстрактную операцию по созданию отдельного объекта класса, которая является общедоступной и не содержит формальных параметров, запись +нарисовать(форма: Многоугольник = прямоугольник, цвет_заливки: Color = (О, О, 255)) – может обозначать операцию по изображению на экране монитора прямоугольной области синего цвета, если не указываются другие значения в качестве аргументов данной операции.

(список–параметров) содержит необязательные аргументы, синтаксис которых совпадает с синтаксисом атрибутов;

< выражение типа возвращаемого значения > является необязательной спецификацией и зависит от конкретного языка программирования;

{строка-свойство} показывает значения свойств, которые применяются к данной операции.

Например, запись запросить_Счет_Клиента(номер_счета:Integer) – обозначает операцию по установлению наличия средств на текущем счете клиента банка. При этом аргументом данной операции является номер счета клиента, который записывается в виде целого числа (например, «123456»).

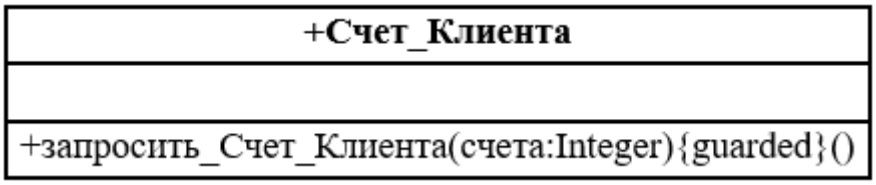


Рисунок 106 – Обозначение операции «запросить_Счет_Клиента»

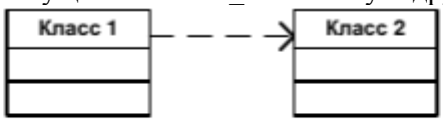
Квантор видимости для операции может быть опущен. В этом случае его отсутствие означает, что видимость операции не указывается.

3.4. Отношения между классами

Классы на диаграмме связываются различными типами отношений. При этом совокупность типов таких отношений фиксирована в языке UML и предопределена их семантикой.

3.4.1. Отношение зависимости

Отношением зависимости (dependency relationship) называют связь по использованию, когда изменение в спецификации одного класса может повлиять на поведение другого. Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели. Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов, направленной к той сущности, от которой зависит данная сущность. Например, некая сущность Класс_2 использует другую сущность Класс_1 (рис. 6).



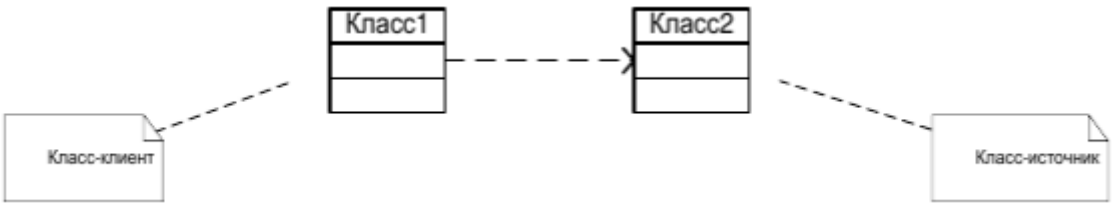


Рисунок 107 – Обозначение отношения зависимости

В качестве класса-клиента и класса-источника зависимости может выступать множество элементов модели. В этом случае одна линия со стрелкой, выходящая от источника зависимости, расщепляется в некоторой точке на несколько отдельных линий, каждая из которых имеет отдельную стрелку для класса-клиента.

Например, если функционирование сущности Класс_С зависит от особенностей реализации сущностей Класс_А и Класс_Б, то данная зависимость может быть изображена следующим образом (рис. 7).

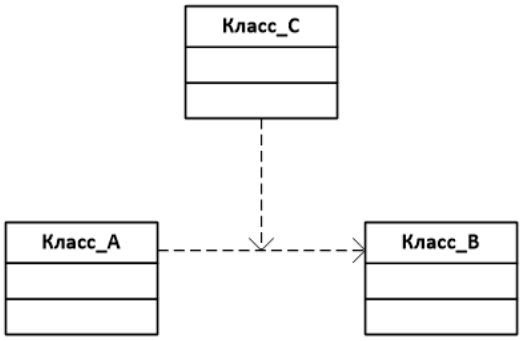


Рисунок 108 – Обозначение зависимости между классом-клиентом (Класс_С) и классами-источниками (Класс_А и Класс_В)

3.4.2. Отношение ассоциации

Ассоциацией (association relationship) называется структурная связь, показывающая, что объекты одного класса некоторым образом связаны с объектами другого или того же самого класса.

Ассоциация может отображаться графически линией со стрелкой (маркером в виде треугольника), показывающей направление следования классов и кратность – количество объектов, связанных отношением. Отсутствие стрелки рядом с именем ассоциации означает, что порядок следования классов в рассматриваемом отношении не определен (рис. 8).

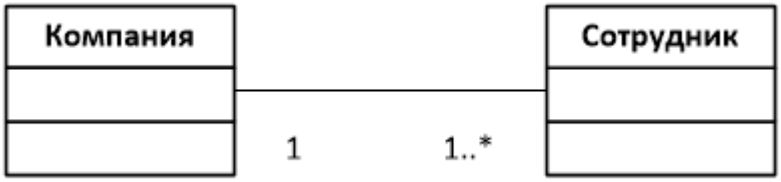


Рисунок 109 – Обозначение отношения ассоциации

Так, в примере на рис. 8 кратность «1» для класса «Компания» означает, что каждый сотрудник может работать только в одной компании. Кратность «1..*» для класса «Сотрудник» означает, что в каждой компании могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено.

Специальной формой или частным случаем отношения ассоциации является отношение агрегации, которое, в свою очередь, тоже имеет специальную форму – отношение композиции (см. пункт 3.4.4).

3.4.3. Отношение агрегации

Отношение агрегации (generalization relationship) имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.

Данное отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа «часть – целое».

Это отношение по своей сути описывает декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции, если в этом возникнет необходимость в последующем.

Так, автомобиль состоит из кузова, двигателя, трансмиссии и т.п., а в состав приемопередающего устройства входят передатчик, приемник и антенно-фидерное устройство.

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой геометрическую фигуру – ромб. Этот ромб указывает на тот из классов, который представляет собой «целое» (рис. 9).

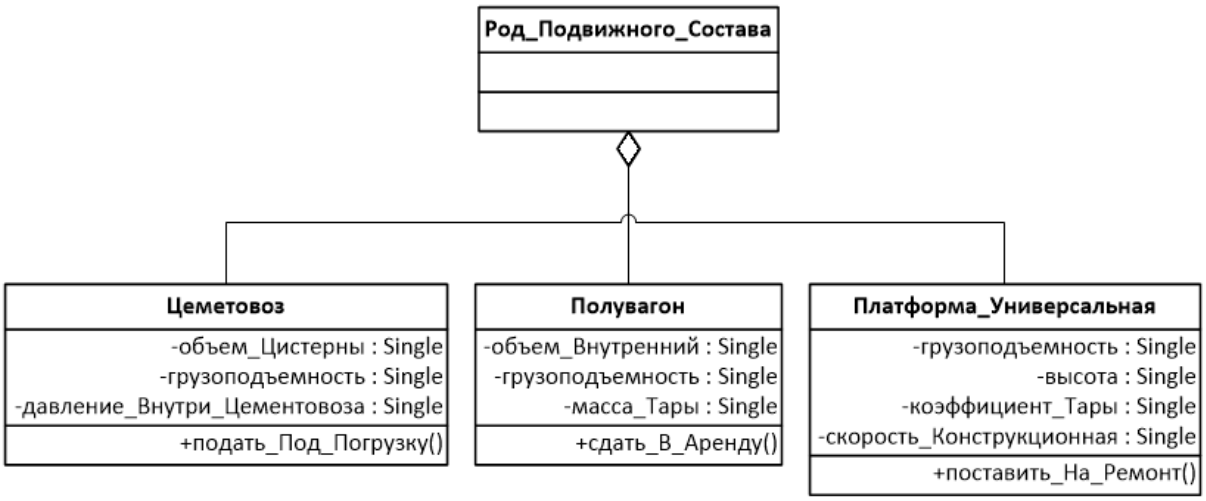


Рисунок 110 – Обозначение отношения агрегации

Примером отношения агрегации может служить деление класса Аналитическая_информация на составные части: Отчет_по_грузу, Отчет_по_контейнерам, Отчет_по_тарифам (рис. 10).

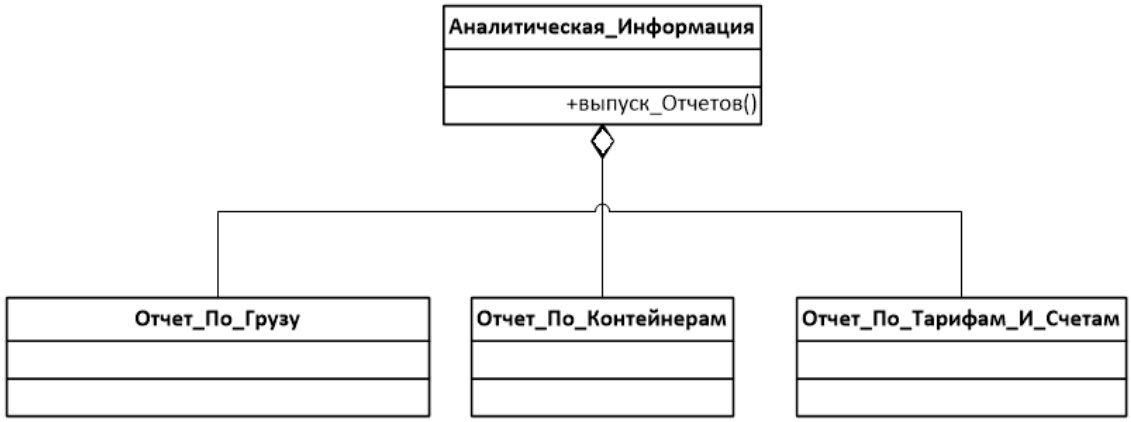


Рисунок 111 – Пример отношения агрегации

Отношение агрегации обладает кратностью. Так, класс Система_обеспечения_безопасности_объектов может содержать содержит одну подсистему Сопровождение_грузов, которая в свою очередь может содержать, например, четыре класса Охрана_вооруженная, каждый из которых может принадлежать лишь одному классу Сопровождение_грузов (рис. 11).

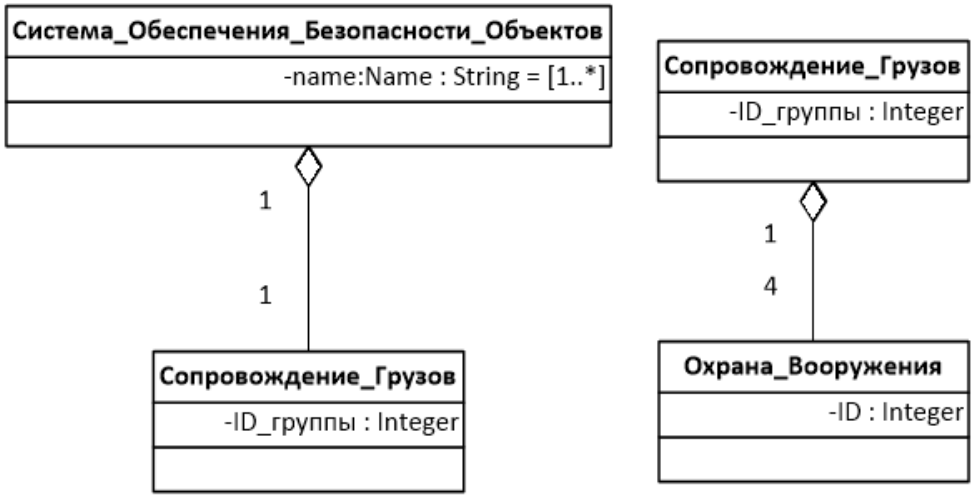


Рисунок 112 – Пример обозначения кратности отношения агрегации

3.4.4. Отношение композиции

Отношение композиции (realization relationship) служит для выделения специальной формы отношения «часть-целое», при которой составляющие части в некотором смысле находятся внутри целого.

Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, т. е. с уничтожением целого уничтожаются и все его составные части.

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой класс-композицию или «целое» (рис. 12).

Рисунок 113 – Обозначение отношения композиции

Применительно к классу Заказ_на_перевозку_грузов отношение композиции может иметь следующий вид (рис. 13).

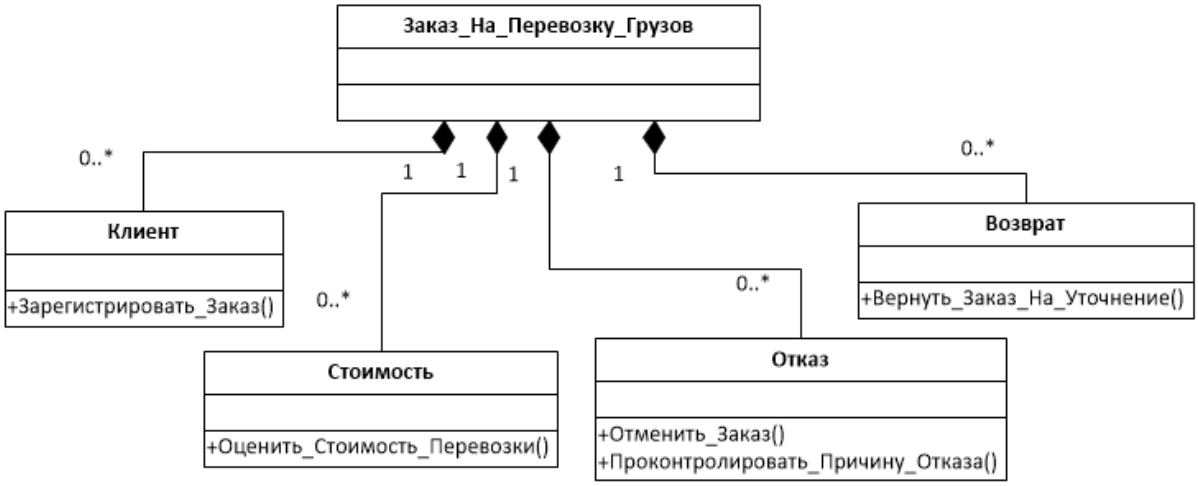


Рисунок 114 – Обозначение на диаграмме отношения композиции

3.4.4. Отношение обобщения

Отношение обобщения (генерализация) является обычным таксономическим отношением между более общим элементом (класс-предок) и более частным или специальным элементом (класс-потомок).

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка.

На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов, направленной на более общий класс (класс-предок или суперкласс) от более специального класса (класса-потомка или подкласса) (рис. 14).

Как правило, на диаграмме может указываться несколько линий для одного отношения обобщения, что отражает его таксономический характер. В этом случае более общий класс разбивается на подклассы одним отношением обобщения, например, так, как показано на рис. 14.

В этом случае данные отдельные линии изображаются сходящимися к единственной стрелке, имеющей с ними общую точку пересечения. Родительский Класс Отчет_По_Заказам_На_Перевозку имеет три потомка Отчет_По_Количеству_Заказов, Отчет_По_Клиентам, Отчет_За_Период, которые наследуют структуру и поведение родительского класса.

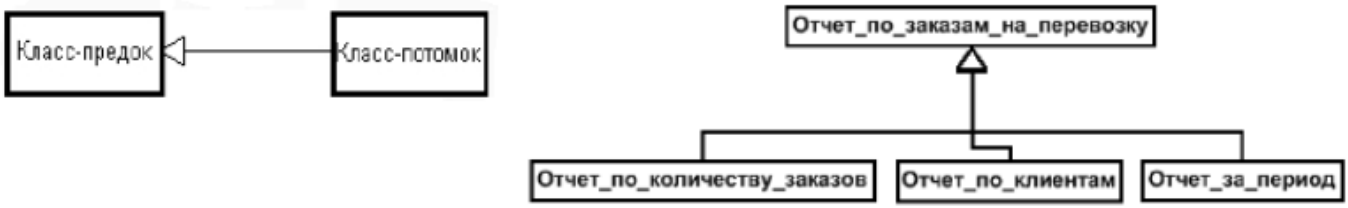


Рисунок 115 – Обозначение на диаграмме отношения обобщения

Для связей обобщения язык UML содержит ограничения. В большинстве случаев ограничение размещается рядом с элементом и заключается в фигурные скобки, например {complete}.

В качестве ограничений могут быть использованы следующие ключевые слова языка UML:

1. {complete} означает, что в данном отношении обобщения специфицированы все классы-потомки, и других классов-потомков у данного класса-предка быть не может.

Например, класс Клиент_банка является предком для двух классов: Физическое_лицо и Компания, и других классов-потомков он не имеет.

На соответствующей диаграмме классов это можно указать явно, записав рядом с линией обобщения данную строку-ограничение (рис. 15).

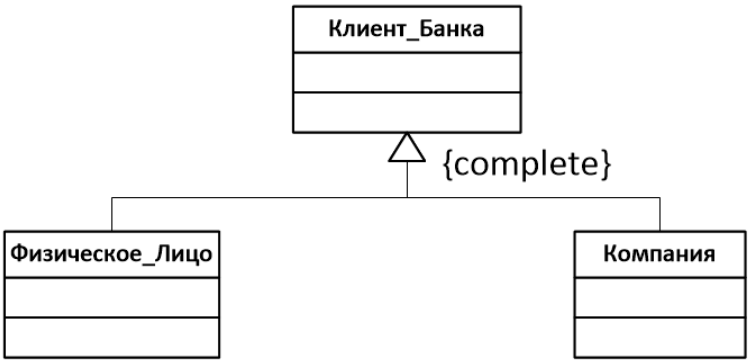


Рисунок 116 – Обозначение ограничения {complete} отношения обобщения

2. {incomplete} означает тот факт, что на диаграмме указаны в обобщении не все классы-потомки. В последующем, возможно, восполнить их перечень, не изменяя уже построенную диаграмму.

3. {disjoint} означает тот факт, что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов.

В приведенном выше примере это условие также выполняется, поскольку предполагается, что никакое конкретное физическое лицо не может являться одновременно и конкретной компанией. В этом случае рядом с линией обобщения можно записать данную строку-ограничение.

4. {overlapping} означает, что отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам.

Например, класс Транспорт может быть специализирован путем создания подклассов Наземный_Транспорт и Водный_Транспорт, автомобиль – амфибия относится к обоим классам.

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

10.Запустите MS Visio.

11.На экране выбора шаблона выберите категорию *Программы и БД* и в ней элемент *Схема модели UML*. Нажмите кнопку *Создать* в правой части экрана.

12.Окно программы примет вид, подобный рис. 16.

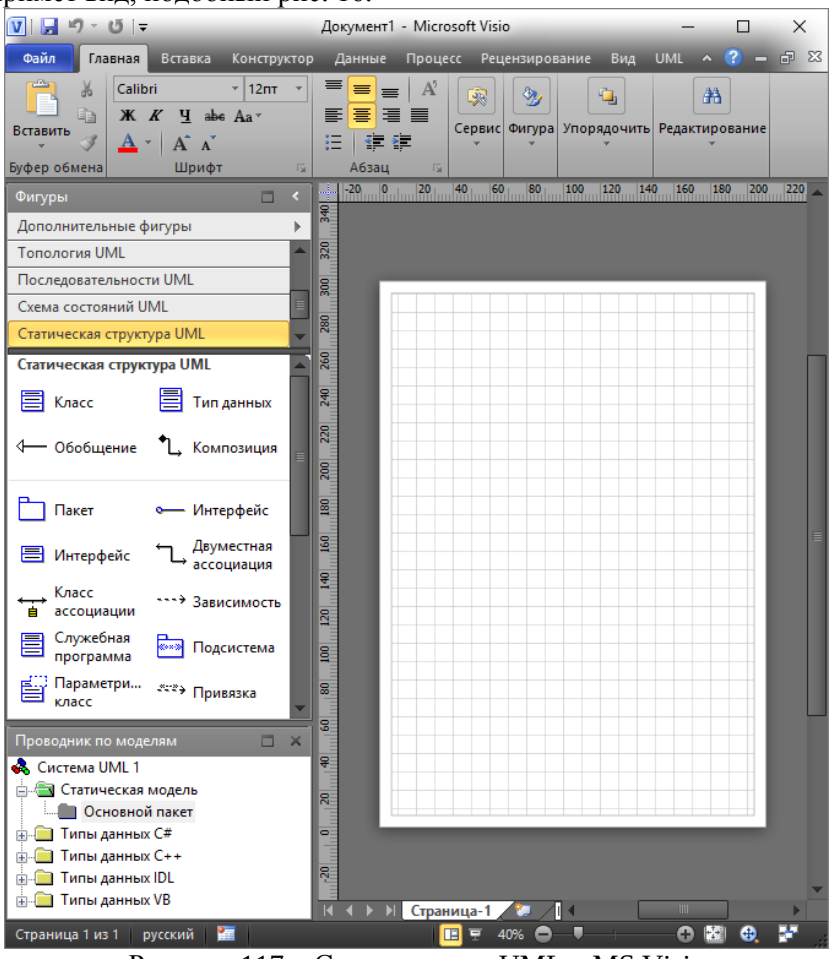


Рисунок 117 – Схема модели UML в MS Visio

13.Ознакомьтесь с элементами графического интерфейса и найдите обязательные панели инструментов **Фигуры**, содержащие категории **Деятельность UML, Взаимодействия UML, Компоненты UML, Топология UML, Последовательности UML, Схема состояний UML, Статическая структура UML, Сценарий выполнения UML, Проводник по моделям**, содержащий иерархическую структуру объектов Системы UML1, Рабочую область, ярлык Страница_1, горизонтальную и вертикальную линейки. (рис. 17).

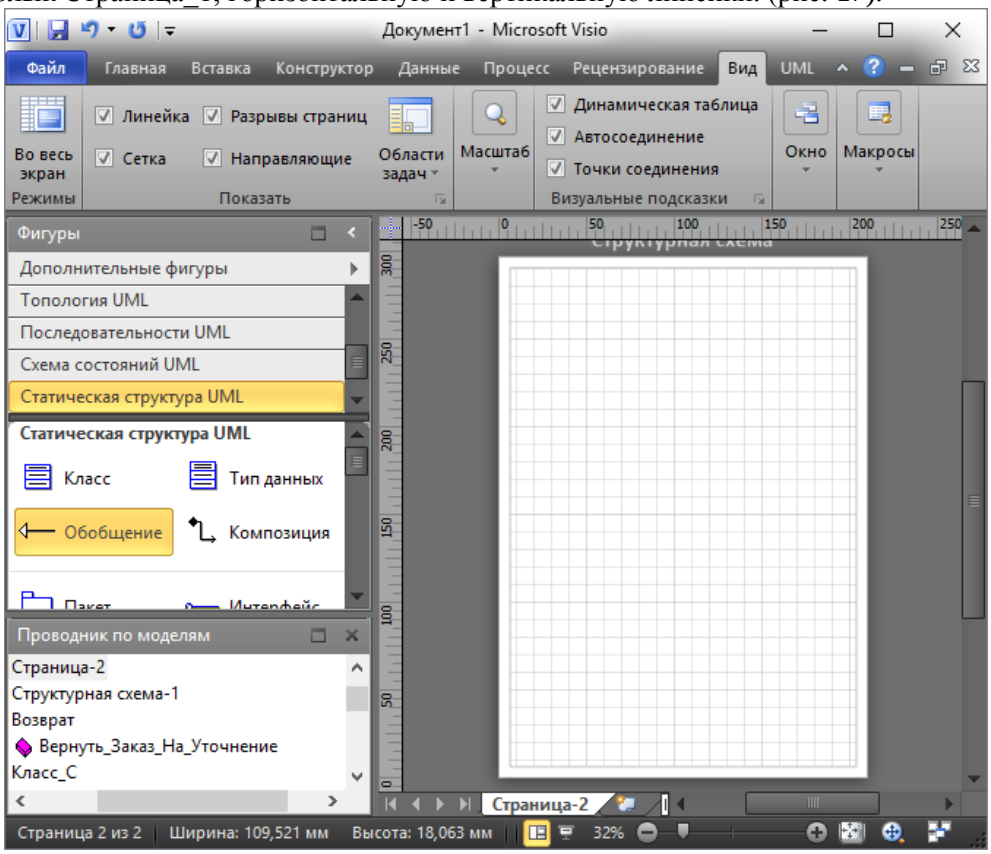


Рисунок 118 – Добавление фигур UML

14.Установите следующие параметры страницы: **Ориентация** – Альбомная, **Автоподбор размера** – выключен, **Имя страницы** – *Диаграмма классов для системы продажи товаров по каталогу.*

15.Перейдите в категорию **Статическая структура UML**, ознакомьтесь с содержимым этой категории и найдите элементы: Класс, Пакет, Подсистема, Интерфейс, Метакласс, Двусторонняя ассоциация, Обобщение, Композиция, Примечание, Ограничение и др.

16.Создайте поэтапно статическую структуру классов UML, с помощью которой может быть сформирована некоторая функциональная часть системы, например, *Система продажи товаров по каталогу*. Для чего:

– Выберите структурные элементы (идентифицируйте классы), участвующие в организации продаж, например, *Продавец, Товар, Заказ, Заказ_Оплата, Клиент, Корпоративный_Клиент, Частный_Клиент* и создайте предварительный вариант совокупности классов с указанием имен (один из возможных вариантов представлен на рис. 18).

– Установите для каждого класса атрибуты в соответствии с перечнем и содержательным описанием бизнес-процессов:

например, для класса *Продавец* в качестве атрибутов могут выступать данные: *фамилия, имя, отчество, телефон*. В данном случае все атрибуты видимы, принадлежат основному пакету *Продавец* (рис. 19).

для класса *Товар* в качестве атрибутов могут выступать данные: тип, марка, артикул (рис. 20).

Рисунок 119 – Формирование статической структуры объектов диаграммы

Рисунок 120 – Описание атрибутов класса Продавец

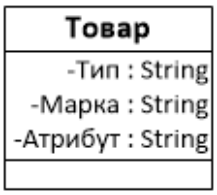


Рисунок 121 – Описание атрибутов класса Товар

для класса *Заказ* в качестве атрибутов могут выступать данные: количество, цена, статус, а в качестве операций – сформировать заказ.

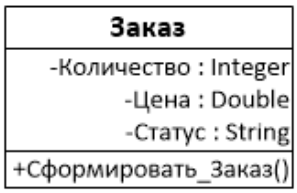


Рисунок 122 – Описание атрибутов и операций класса Заказ

для класса *Заказ_Оплата* в качестве атрибутов могут выступать данные: дата получения, проплачен, номер, цена, а в качестве операций – отправить, закрыть.

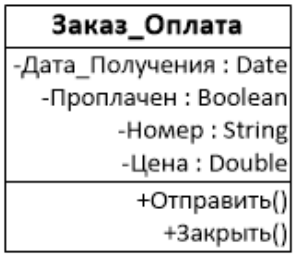


Рисунок 123 – Описание атрибутов и операций класса Заказ_Оплата

для класса *Клиент* в качестве атрибутов могут выступать данные: имя, адрес, а в качестве операций – кредитный рейтинг.

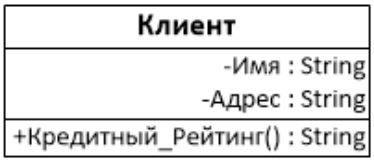


Рисунок 124 – Описание атрибутов и операций класса Клиент

для класса *Корпоративный_Клиент* в качестве атрибутов могут выступать данные: контактное имя, кредитный рейтинг, кредитный лимит, а в качестве операций – сделать, напоминание, счет за месяц.

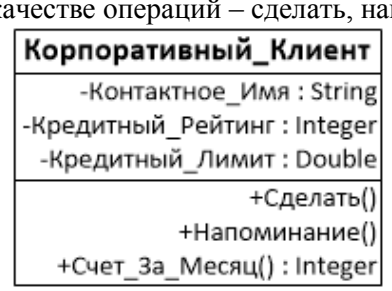


Рисунок 125 – Описание атрибутов и операций класса Корпоративный_Клиент

для класса *Частный_Клиент* в качестве атрибутов могут выступать данные: номер кредитной карты.

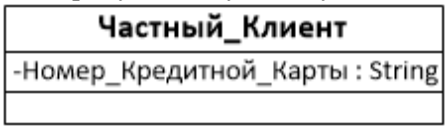


Рисунок 126 – Описание атрибутов класса Частный_Клиент

для класса *Вариант_Оплаты* в качестве атрибутов могут выступать данные: тип оплаты, а в качестве операций – выбор варианта оплаты.

Рисунок 127 – Описание атрибутов и операций класса Вариант_Оплаты

для класса *Каталог_Товаров* в качестве атрибутов могут выступать данные: тип, марка, артикул, а в качестве операций – проверить наличие.

Рисунок 128 – Описание атрибутов и операций класса Каталог_товаров

для класса *Склад* в качестве атрибутов могут выступать данные: товар, наличие, количество, а в качестве операций – Проверить наличие.

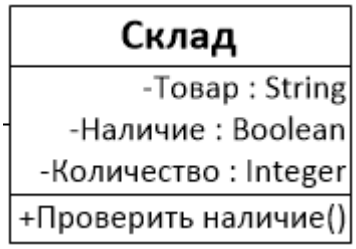


Рисунок 129 – Описание атрибутов и операций класса Склад

– Убедитесь, что все элементы наполнены адекватным содержанием и расположите все структурные элементы диаграммы наиболее оптимально на странице для установления отношений между ними.

В качестве примера на рис. 26 показан набор классов, описывающих реализацию системы продаж товаров по каталогу. Акцент сделан на классе *Клиент*, с которым связан класс *Заказ_Оплата* посредством двусторонней ассоциации «один-ко-многим», *Вариант_Оплаты* – двусторонней ассоциацией «один-к-одному» и классы *Корпоративный_Клиент* и *Частный_Клиент* посредством отношения обобщения. Классы *Заказ_Оплата* и *Товар* связаны с классом *Заказ* посредством двусторонней ассоциации «один-ко-многим». Класс *Товар* связан с классом *Продавец* двусторонней ассоциацией «многие-ко-многим» и классом *Каталог_Товаров* двусторонней ассоциацией «один-ко-многим». Класс *Каталог_Товаров* связан посредством двусторонней ассоциации «многие-ко-многим» с классом *Склад.*

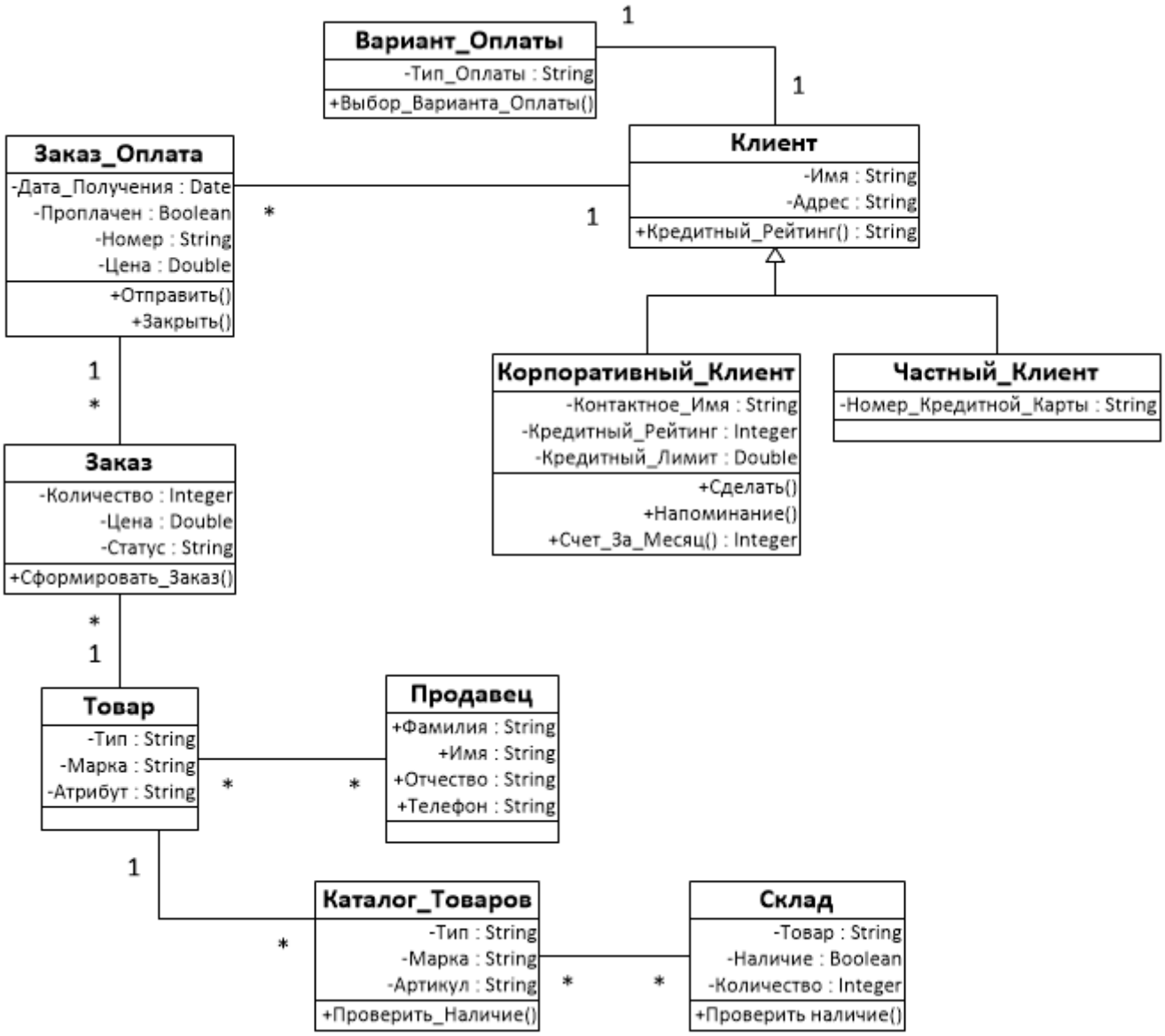


Рисунок 130 – Фрагмент диаграммы классов, описывающей реализацию систем продаж товаров по каталогу

17.Создайте новую страницу с именем *Диаграмма классов учета клиентов*, и установите следующие опции: **Ориентация** – Альбомная, **Автоподбор размера** – выключен.

18.Идентифицируйте классы учета клиентов, осуществляющих заказы и создайте диаграмму классов с указанием их имен, атрибутов, операций, например, так как показано на рис. 30.

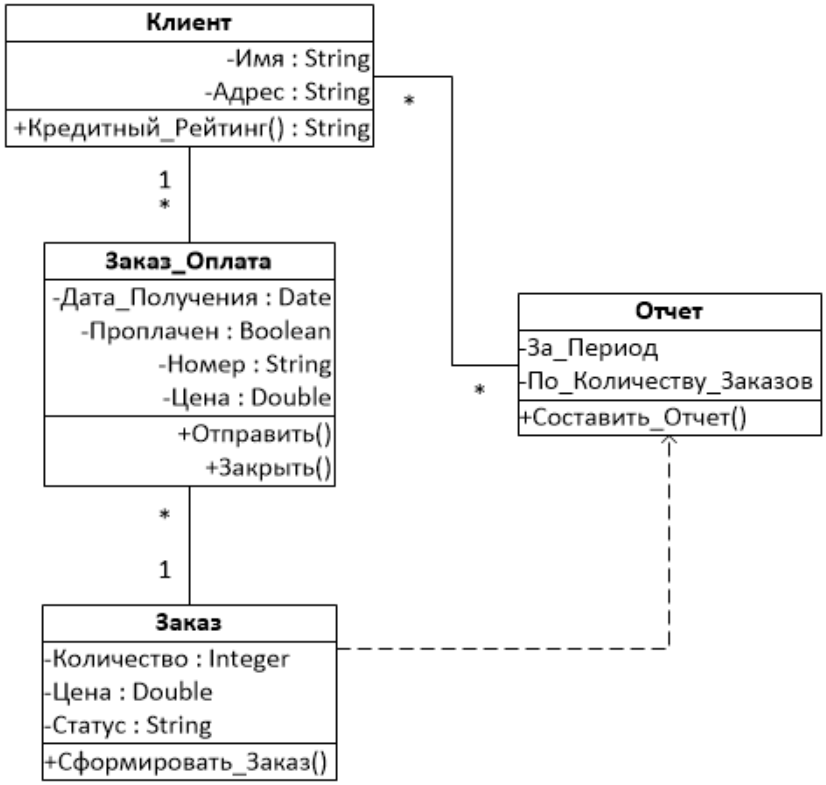


Рисунок 131 – Фрагмент диаграммы классов, описывающей процесс формирования отчетности

5. Задание

Построить диаграмму классов в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Каково назначение диаграммы классов?
2. Назовите основные элементы диаграммы классов.
3. Какие виды связей доступны в диаграмме классов?
4. Для чего используется каждый вид связи?
5. Как создать диаграмму классов в VISIO?

Практическая работа №9

Построение диаграмм состояний на языке UML с помощью MS Visio

1. Цель работы

Целью работы является изучение основ создания диаграмм состояний на языке UML, получение навыков построения диаграмм состояний, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

2. Задачи

Основными задачами практической работы являются:

– ознакомиться с теоретическими вопросами построения диаграмм состояний на языке UML;
– ознакомиться с теоретическими вопросами построения диаграмм состояний с помощью MS Visio.

3. Краткие теоретические сведения

Диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее – одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта. При этом изменение состояния объекта может быть вызвано внешними воздействиями со стороны других объектов или извне. Именно для описания реакции объекта на подобные внешние воздействия и используются диаграммы состояний.

Диаграмма состояний описывает возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

Хотя диаграммы состояний чаще всего используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Понятие автомата в контексте UML обладает довольно специфической семантикой, основанной на теории автоматов. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели.

Рисунок 132 – Пример диаграммы состояний для технического устройства типа «Компьютер»

Состояние

Под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.

Рисунок 133 – Графическое обозначение состояния

Секция «Список внутренних действий» содержит перечень внутренних действий или деятельностей, которые выполняются в процессе нахождения моделируемого элемента в данном состоянии. Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

<метка-действия '/' выражение-действия>

Метка действия указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия:

– ***entry*** – эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент входа в данное состояние (входное действие);

– ***exit*** – эта метка указывает на действие, специфицированное следующим за ней выражением действия, которое выполняется в момент выхода из данного состояния (выходное действие);

– ***do*** – эта метка специфицирует выполняющуюся деятельность («doactivity»), которая выполняется в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не закончится вычисление, специфицированное следующим за ней выражением действия. В последнем случае при завершении события генерируется соответствующий результат;

– ***include*** – эта метка используется для обращения к подавтомату, при этом следующее за ней выражение действия содержит имя этого подавтомата.

Пример: Аутентификация входа

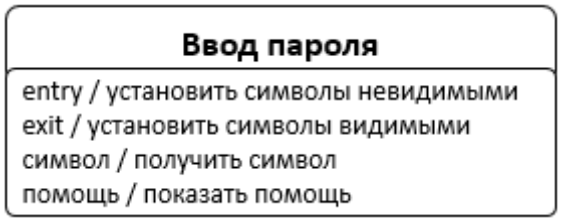


Рисунок 134 – Состояние для ввода пароля

Начальное и конечное состояния

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). 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.

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосостояния). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени.

Рисунок 135 – Графическое изображение начального и конечного состояний

Переход

Простой переход (simpletransition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. В этом случае говорят, что переход срабатывает, Или происходит срабатывание перехода. До срабатывания перехода объект находится в предыдущем от него состоянии, называемым исходным состоянием, или в источнике (не путать с начальным состоянием – это разные понятия), а после его срабатывания объект находится в последующем от него состоянии (целевом состоянии).

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (doactivity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события. Кроме того, на переходе могут указываться действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое. Срабатывание перехода может зависеть не только от наступления некоторого события, но и от выполнения определенного условия, называемого сторожевым условием. Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение «истина».

На диаграмме состояний ***переход изображается*** сплошной линией со стрелкой, которая направлена в целевое состояние. Каждый переход может помечен строкой текста, которая имеет следующий общий формат:

<сигнатура события>'['<сторожевое условие>']' <выражение действия>

Событие

Событие представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. Про события говорят, что они «происходят», при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

События играют роль стимулов, которые инициируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. Имя события идентифицирует каждый отдельный переход на диаграмме состояний и может содержать строку текста, начинающуюся со строчной буквы.

Сторожевое условие

Сторожевое условие (guardcondition), если оно есть, всегда записывается в прямых скобках после события и представляет собой некоторое булевское выражение.

Если сторожевое условие принимает значение «истина», то соответствующий переход может сработать, в результате чего объект перейдет в целевое состояние. Если же сторожевое условие принимает значение «ложь», то переход не может сработать, и при отсутствии других переходов объект не может перейти в целевое состояние по этому переходу. Однако вычисление истинности сторожевого условия происходит только после возникновения ассоциированного с ним события, инициирующего соответствующий переход.

В общем случае из одного состояния может быть несколько переходов с одним и тем же событием-триггером. При этом никакие два сторожевых условия не должны одновременно принимать значение «истина». Каждое из сторожевых условий необходимо вычислять всякий раз при наступлении соответствующего события.

Пример

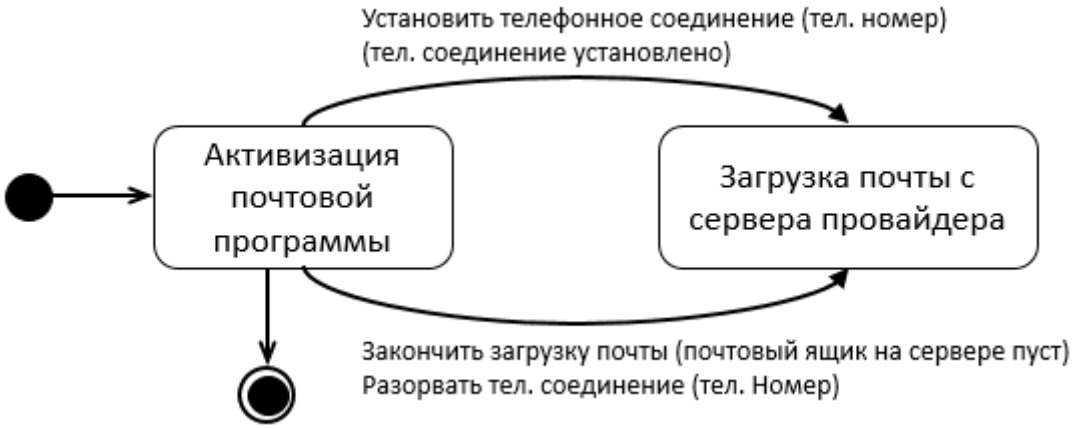


Рисунок 136 – Диаграмма состояний для моделирования почтовой программы клиента

Составное состояние и подсостояние

Составное состояние (compositestate) – такое сложное состояние, которое состоит из других вложенных в него состояний. Последние будут выступать по отношению к первому как подсостояния (substate).

Рисунок 137 – Графическое изображение составного состояния

Последовательные подсостояния (sequential substates) используются для моделирования такого поведения объекта, во время которого в каждый момент времени объект может находиться в одном и только одном подсостояний. Поведение объекта в этом случае представляет собой последовательную смену подсостояний, начиная от начального и заканчивая конечным подсостояниями. Хотя объект продолжает находиться в составном состоянии, введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты его внутреннего поведения.

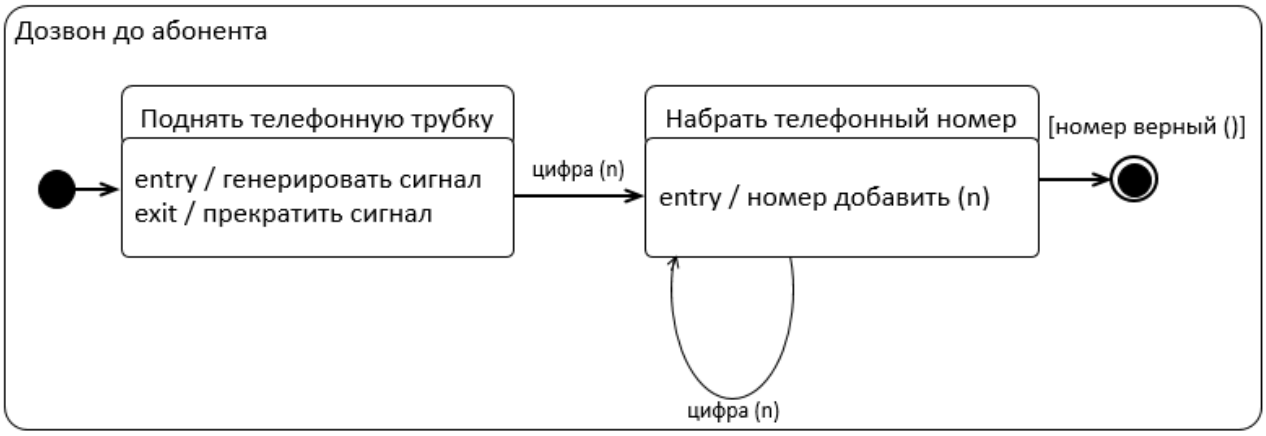


Рисунок 138 – Пример составного состояния с двумя вложенными последовательными подсостояниями

Параллельные подсостояния (concurrentsubstates) позволяют специфицировать два и более подавтомата, которые могут выполняться параллельно внутри составного события. Каждый из подавтоматов занимает некоторую область (регион) внутри составного состояния, которая отделяется от остальных горизонтальной пунктирной линией. Если на диаграмме состояний имеется составное состояние с вложенными параллельными подсостояниями, то объект может одновременно находиться в каждом из этих подсостояний.

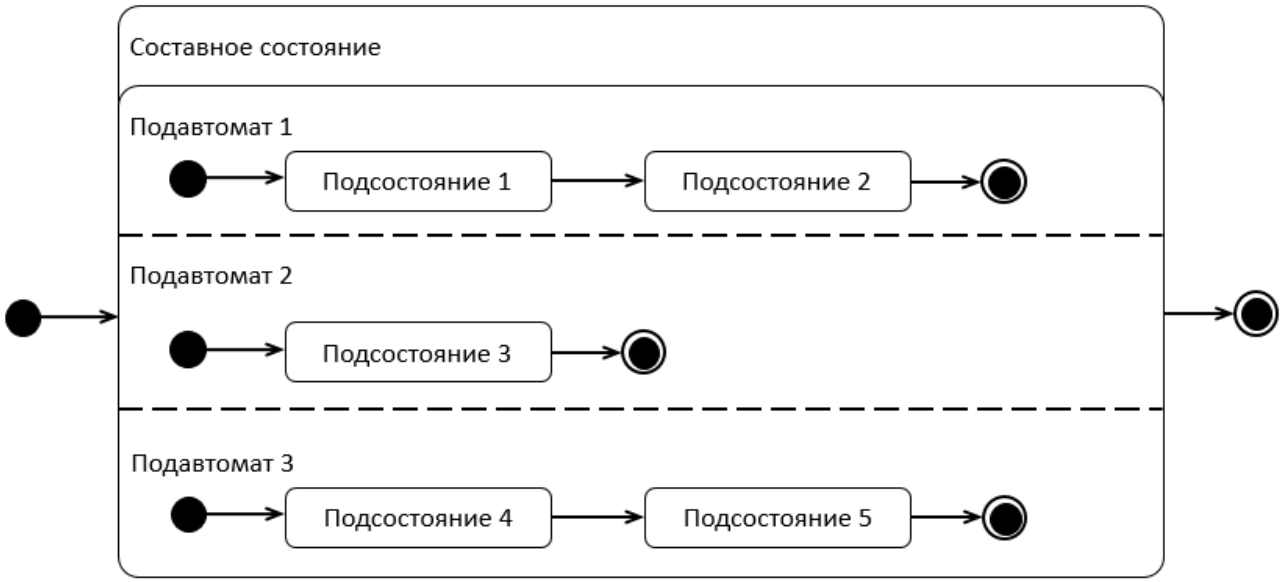


Рисунок 139 – Графическое изображение составного состояния с вложенными параллельными подсостояниями

Синхронизирующие состояния

Поведение параллельных подавтоматов независимо друг от друга, что позволяет реализовать многозадачность в программных системах. Однако в отдельных случаях может возникнуть необходимость учесть в модели синхронизацию наступления отдельных событий. Для этой цели в языке UML имеется специальное псевдосостояние, которое называется синхронизирующим состоянием.

Синхронизирующее состояние (synch state) обозначается небольшой окружностью, внутри которой помещен символ звездочки "*". Оно используется совместно с переходом-соединением или переходом-ветвлением для того, чтобы явно указать события в других подавтоматах, оказывающие непосредственное влияние на поведение данного подавтомата.

Для иллюстрации использования синхронизирующих состояний рассмотрим упрощенную ситуацию с моделированием процесса постройки дома. Предположим, что постройка дома включает в себя строительные работы (возведение фундамента и стен, возведение крыши и отделочные работы) и работы по электрификации дома (подведение электрической линии, прокладка скрытой электропроводки и установка осветительных ламп). Очевидно, два этих комплекса работ могут выполняться параллельно, однако между ними есть некоторая взаимосвязь.

В частности, прокладка скрытой электропроводки может начаться лишь после того, как будет завершено возведение фундамента и стен. А отделочные работы следует начать лишь после того, как будет закончена прокладка скрытой электропроводки. В противном случае отделочные работы придется проводить повторно. Рассмотренные особенности синхронизации этих параллельных процессов учтены на соответствующей диаграмме состояний с помощью двух синхронизирующих состояний.

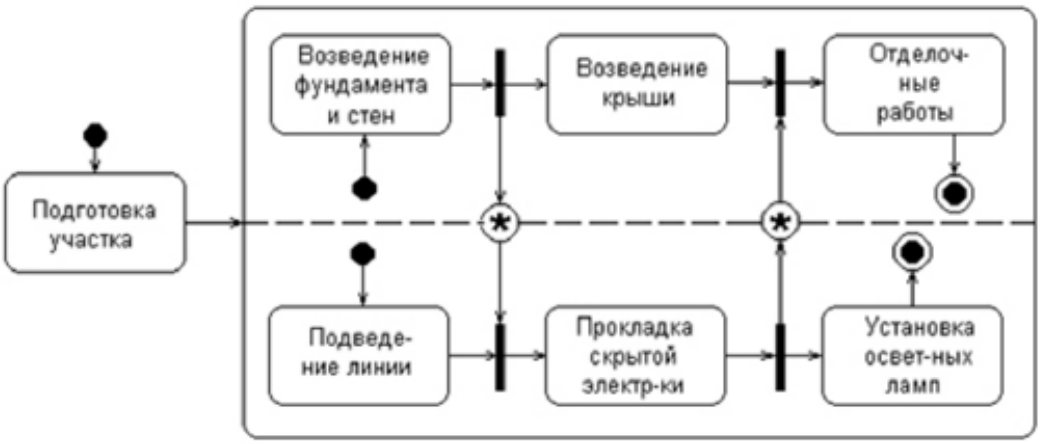


Рисунок 140 – Диаграмма состояний для примера со строительством дома

Примеры диаграмм состояний изображены на рисунках 10-12.

Рисунок 141 – Диаграмма состояний для моделирования запроса данных из БД

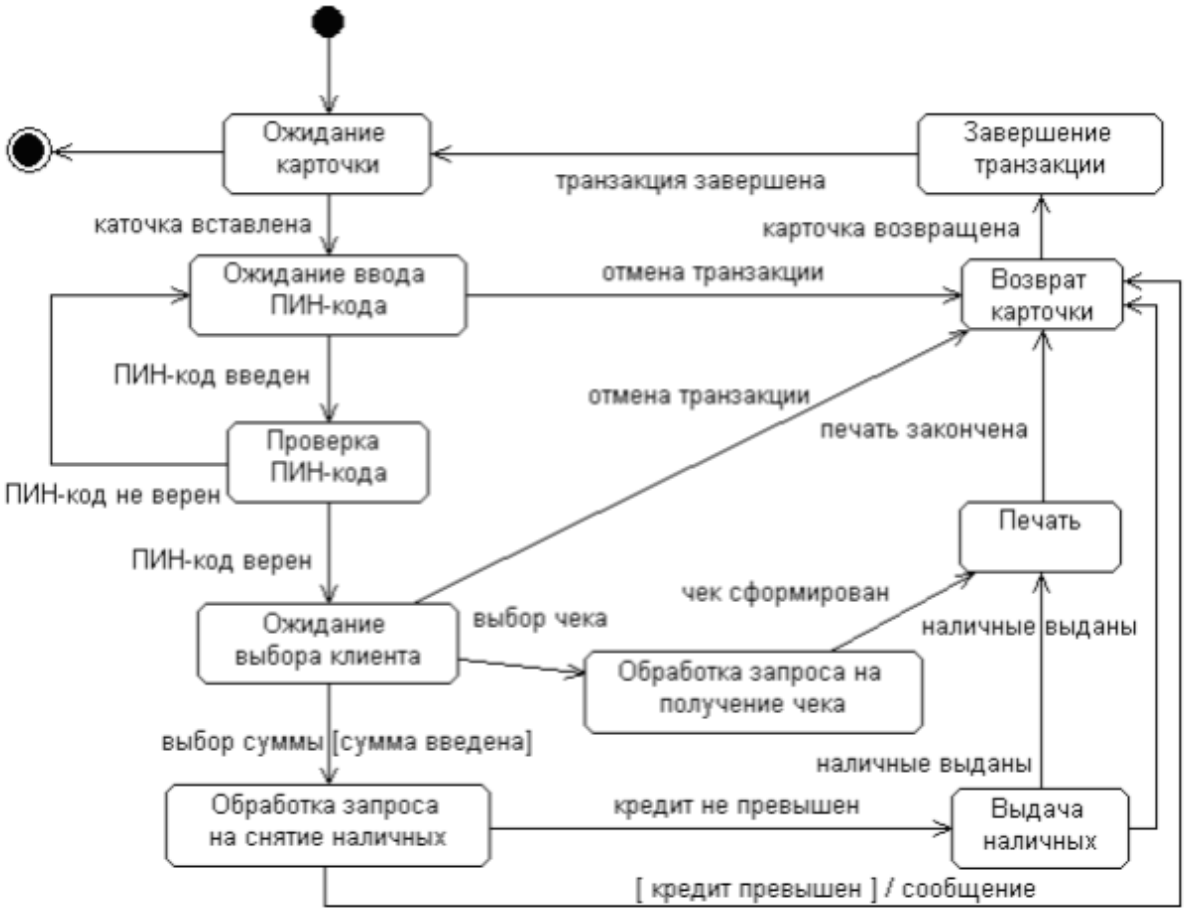


Рисунок 142 – Диаграмма состояний для моделирования поведения банкомата

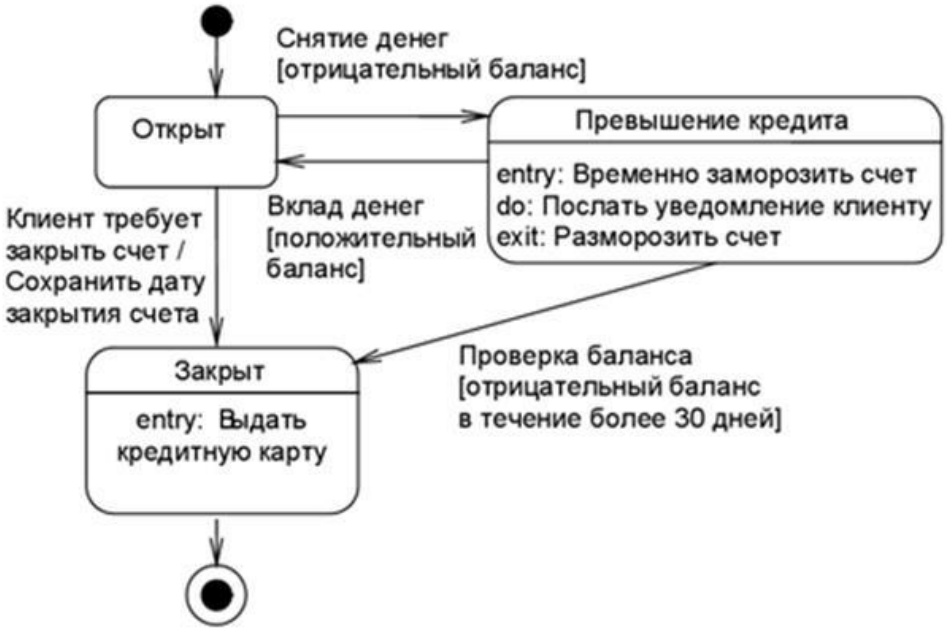


Рисунок 143 – Диаграмма состояний моделирования деятельности сотрудника банка

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

19.Запустите MS Visio.

20.На экране выбора шаблона выберите категорию *Программы и БД* и в ней элемент *Схема модели UML*. Нажмите кнопку *Создать* в правой части экрана.

21.Окно программы примет вид, подобный рис. 13.

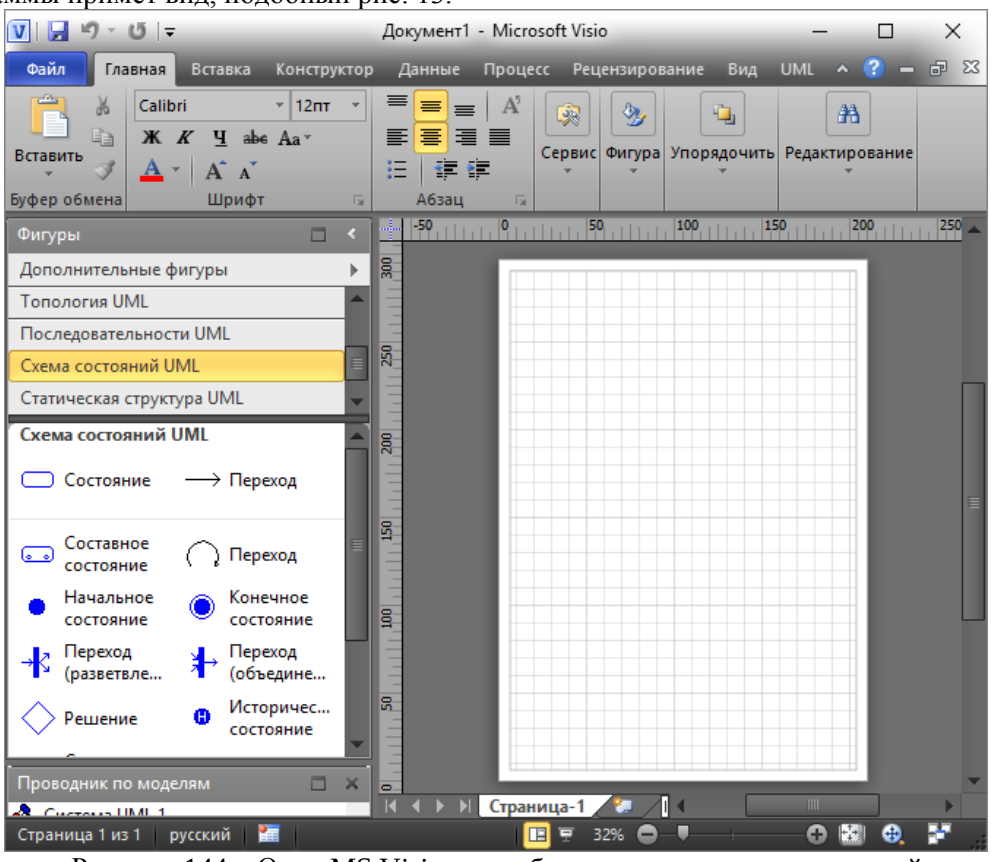


Рисунок 144 – Окно MS Visio для работы с диаграммами состояний

Клиент оформляет заказ. Класс Заказ имеет атрибут Статус. Проследим динамику движения заказов в системе с помощью ***диаграммы состояний, составленной для класса Заказ.***

Данные о сформированном заказе поступают продавцу, который проверяет наличие товаров из заказа, проверяет оплату заказа, комплектует его и делает отметку о готовности. После оплаты заказа он выдается клиенту. Продавец делает отметку о том, что заказ выдан.

Если после проверки кредитного рейтинга клиента, он окажется отрицательным, то заказ будет отклонен.

Построим диаграмму состояний для класса Заказ. Для этого, в файле с диаграммой классов, созданной в практическом занятии 8, необходимо проделать следующие действия:

1. Щелкнуть правой кнопкой мыши по *классу Заказ*.
2. В контекстном меню выбрать пункт *Схемы*.
3. Т.к. в настоящее время уже созданных схем нет, нажать кнопку *Создать* и выбрать *Схема состояний*.
4. Переименовать созданный лист в *Схема состояний-Заказ*.
5. Построить диаграмму состояний для класса Заказ в соответствии с рисунком 14.

После формирования заказа он должен быть оплачен. Обработка заказа подразумевает проверку наличия товара и проверку оплаты. Переход в одно из состояний На комплектации, Укомплектован, Выдан означает смену Статуса заказа.

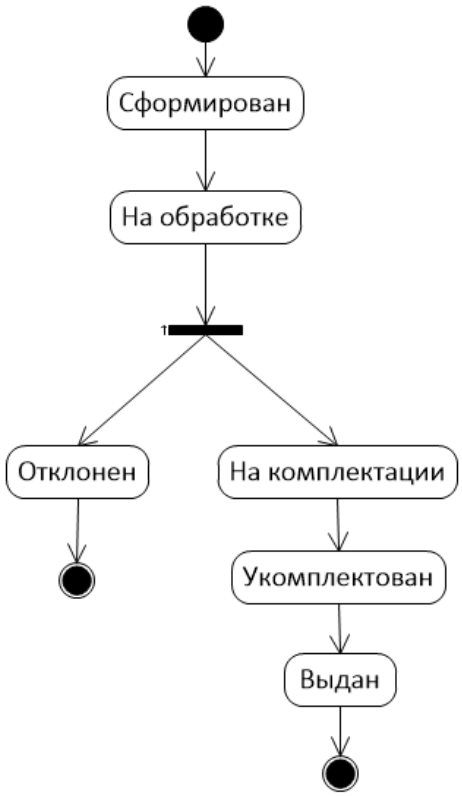


Рисунок 145 – Диаграмма состояний для класса Заказ

Далее опишем с помощью ***диаграммы состояний процесс оплаты заказа клиентом***, которому соответствует класс ЗаказОплата.

Построим диаграмму состояний для проверки оплаты заказа.

Чтобы проверить оплату заказа, необходимо определить, существует ли сам заказ. Результатом проверки оплаты заказа является вывод либо сообщения о произведенной оплате с параметрами (дата оплаты), либо сообщения об ожидании оплаты.

Событием, предшествующим проверке оплаты заказа, является занесение информации о заказе в базу данных заказов.

Чтобы построить диаграмму состояний для класса ЗаказОплаты, необходимо проделать действия, описанные в пунктах 1-4 построения диаграммы состояний для класса Заказ. Полученная диаграмма должна иметь вид, изображенный на рис. 15.

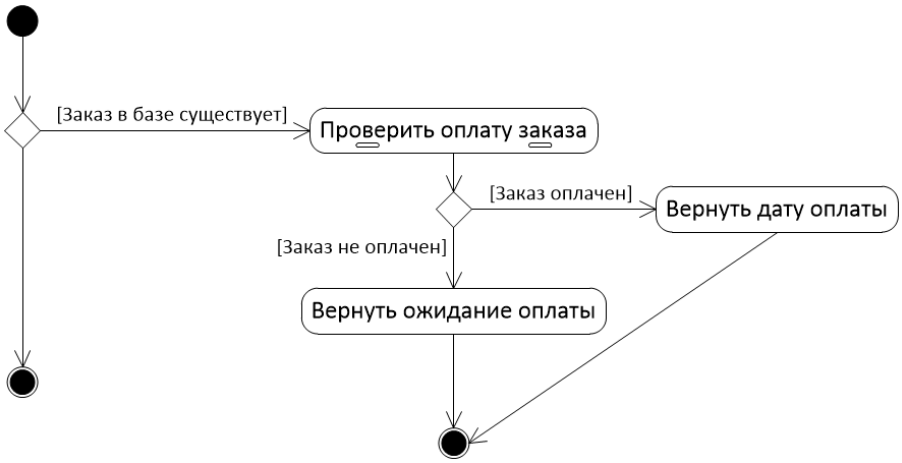


Рисунок 146 – Диаграмма состояний проверки платы заказа

На этой диаграмме есть составное состояние ***«Проверить оплату заказа»***, т.к. оно включает в себя *проверку кредитного рейтинга клиента* и *проверку выбора варианта оплаты клиентом*.

Оплату заказа может произвести только клиент с положительным кредитным рейтингом, поэтому необходимым условием проверки оплаты заказа является проверка кредитоспособности клиента. Если клиент имеет отрицательный кредитный рейтинг, то заказ отклоняется, и на этом дальнейшие события не имеют смысла.

Если кредитный рейтинг клиента положительный, то необходимо проверить, выбрал ли клиент вариант оплаты. Событие, которое переводит систему в состояние ожидания выбора варианта оплаты клиентом, является получение сообщения о кредитоспособности клиента.

Оплата может быть произведена наличными средствами в магазине или с помощью безналичного расчета. В первом случае необходимо договориться с клиентом о дате и времени его прибытия в магазин. Во втором случае необходимо сообщить клиенту о наличии/поступлении товара. Событие, которое переводит систему в состояние ожидания оплаты, является выбор клиентом варианта оплаты.

Соответствующие диаграммы состояний имеют вид (рис. 16 и рис. 17).

Для создания диаграмм состояний, которые входят в состав составного состояния, нужно:

1. Щелкнуть правой кнопкой мыши по Составному состоянию и выбрать пункт Схема.
2. Либо, в Проводнике по моделям выделить название составного состояния и создать новую страницу с помощью кнопки  .

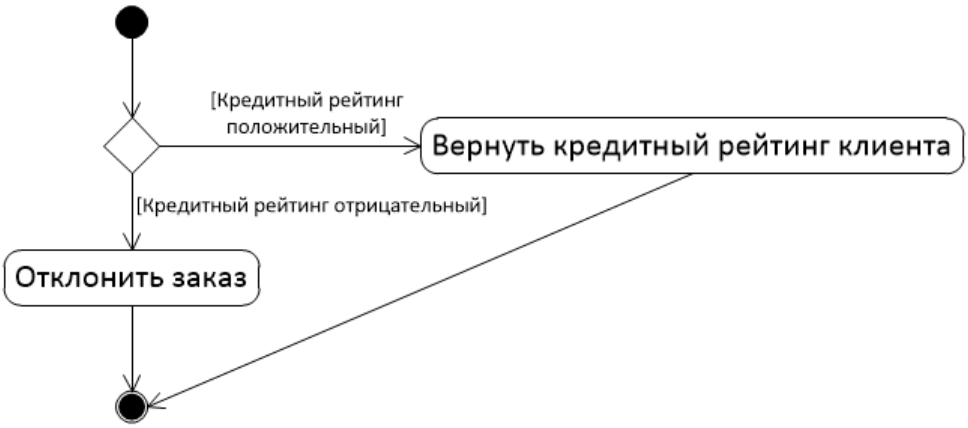


Рисунок 147 – Диаграмма состояний для проверки кредитного рейтинга клиента

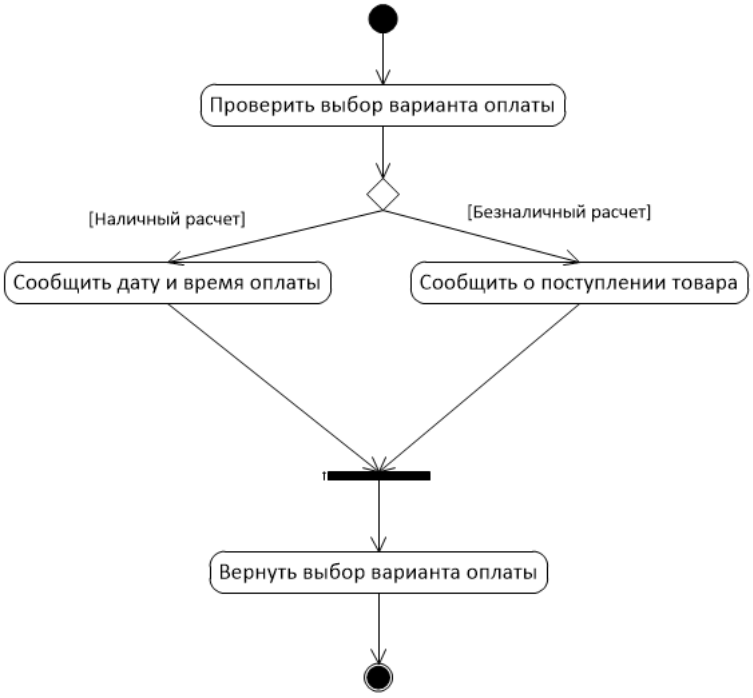


Рисунок 148 – Диаграмма состояний для проверки варианта оплаты

5. Задание

Построить диаграмму состояний в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

Создать диаграммы состояния не менее чем для трех классов, описанных в практическом занятии №8.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Каково назначение диаграммы состояний?
2. Назовите основные элементы диаграммы состояний.
3. Как создать диаграмму состояний в VISIO?
4. В чем отличие диаграммы классов и состояний?

Практическая работа №10

Построение диаграмм деятельности на языке UML с помощью MS Visio

1. Цель работы

Целью работы является изучение основ создания диаграмм деятельности на языке UML, получение навыков построения диаграмм деятельности, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

2. Задачи

Основными задачами практической работы являются:

- ознакомиться с теоретическими вопросами построения диаграмм деятельности на языке UML;
- ознакомиться с теоретическими вопросами построения диаграмм деятельности с помощью MS Visio.

3. Краткие теоретические сведения

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Например, чтобы позвонить по телефону, нам предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций, таких как разборка силового агрегата, снятие генератора и некоторых других.

С увеличением сложности системы строгое соблюдение последовательности выполняемых операций приобретает все более важное значение. Если попытаться заварить кофе холодной водой, то мы можем только испортить одну порцию напитка. Нарушение последовательности операций при ремонте двигателя может привести к его поломке или выходу из строя. Еще более катастрофические последствия могут произойти в случае отклонения от установленной последовательности действий при взлете или посадке авиалайнера, запуске ракеты, регламентных работах на АЭС.

Для моделирования процесса выполнения операций в языке UML используются так называемые **диаграммы деятельности**. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML **деятельность (activity)** представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Состояние действия и деятельности

Состояние деятельности (activity state) – состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определенного времени. Переход из состояния деятельности происходит после выполнения специфицированной в нем ду-деятельности, при этом ключевое слово do в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным.

Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

Состояние деятельности можно представлять себе, как составное состояние, поток управления которого включает только другие состояния деятельности и действий.

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами (рис. 1). Внутри этой фигуры записывается выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

Рисунок 149 – Графическое обозначение состояния действия

Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами (рис. 1а). Если же действие может быть представлено в некотором формальном виде, то целесообразно записать его на том языке программирования, на котором предполагается реализовывать конкретный проект (рис. 1б).

Иногда возникает необходимость представить на диаграмме деятельности некоторое сложное действие, которое, в свою очередь, состоит из нескольких более простых действий. В этом случае можно использовать специальное обозначение так называемого **состояния поддеятельности (subactivity state)**. Такое состояние является графом деятельности и обозначается специальной пиктограммой в правом нижнем углу символа состояния действия (рис. 2). Эта конструкция может применяться к любому элементу языка UML, который поддерживает «вложенность» своей структуры. При этом пиктограмма может быть дополнительно помечена типом вложенной структуры.

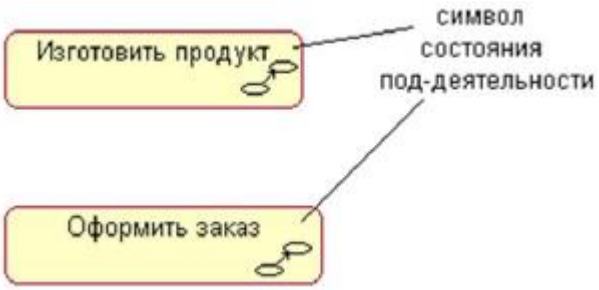


Рисунок 150 – Графическое обозначение состояния поддеятельности

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний (см. практическое занятия №9). При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии. Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное – в ее нижней части.

Переходы

При построении диаграммы деятельности используются только такие переходы, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. Этот переход переводит деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, то сработать может только один из них. Именно в этом случае для каждого из таких переходов должно быть явно записано сторожевое условие в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата. Такая ситуация получила название ветвления, а для ее обозначения применяется специальный символ.

Графически ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста (рис. 3). В этот ромб может входить только одна стрелка от того состояния действия, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа ветвления. Выходящих стрелок может быть две или более, но для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

В качестве **примера** рассмотрим фрагмент известного алгоритма нахождения корней квадратного уравнения. В общем случае после приведения уравнения второй степени к каноническому виду: $a * x * x + b * x + c = 0$ необходимо вычислить его дискриминант. Причем, в случае отрицательного дискриминанта уравнение не имеет решения на множестве действительных чисел, и дальнейшие вычисления должны быть прекращены. При неотрицательном дискриминанте уравнение имеет решение, корни которого могут быть получены на основе конкретной расчетной формулы.

Графически фрагмент процедуры вычисления корней квадратного уравнения может быть представлен в виде диаграммы деятельности с тремя состояниями действия и ветвлением (рис. 3). Каждый из переходов, выходящих из состояния «Вычислить дискриминант», имеет сторожевое условие, определяющее единственную ветвь, по которой может быть продолжен процесс вычисления корней в зависимости от знака дискриминанта. Очевидно, что в случае его отрицательности, мы сразу попадаем в конечное состояние, тем самым завершая выполнение алгоритма в целом.

Рисунок 151 – Фрагмент диаграммы деятельности для алгоритма нахождения корней квадратного уравнения

В рассмотренном примере, как видно из рис. 3, выполняемые действия соединяются в конечном состоянии. Однако это вовсе не является обязательным. Можно изобразить еще один символ ветвления, который будет иметь несколько входящих переходов и один выходящий.

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений. Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем, необходимы графические примитивы для представления параллельных процессов. В языке UML для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления. Таким символом является **прямая черточка.**

Такая черточка изображается отрезком горизонтальной линии, толщина которой несколько шире основных сплошных линий диаграммы деятельности. При этом разделение (concurrent fork) имеет один входящий переход и несколько выходящих (рис. 4а). Слияние (concurrent join), наоборот, имеет несколько входящих переходов и один выходящий (рис. 4б).

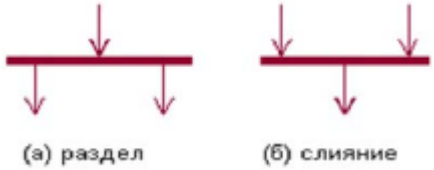


Рисунок 152 – Графическое обозначение разделения и слияния параллельных потоков управления

Для иллюстрации особенностей параллельных процессов выполнения действий рассмотрим **пример** с приготовлением напитка. Достоинство этого примера состоит в том, что он практически не требует никаких дополнительных пояснений в силу своей очевидности (рис. 5).

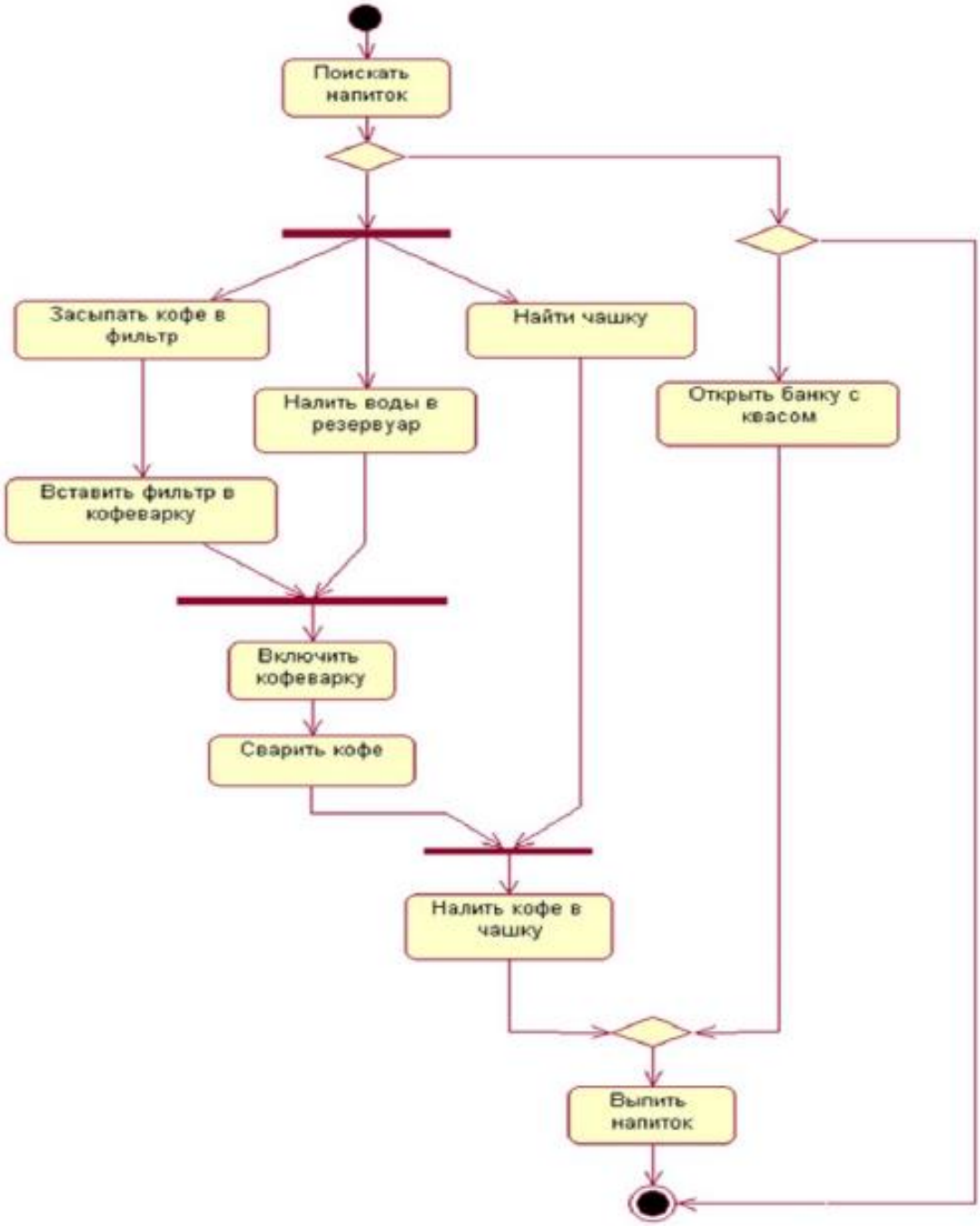


Рисунок 153 – Диаграмма деятельности для примера с приготовлением напитка

Хотя диаграмма деятельности предназначена для моделирования поведения систем, время в явном виде отсутствует на этой диаграмме. Ситуация здесь во многом аналогична диаграмме состояний.

Таким образом, диаграмма деятельности есть не что иное, как специальный случай диаграммы состояний, в которой все или большинство состояний являются действиями или состояниями поддеятельности. А все или большинство переходов являются переходами, которые срабатывают по завершении действий или под-деятельностей в состояниях источниках.

Дорожки

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах. Не менее важная область их применения связана с моделированием бизнес-процессов. Действительно, деятельность любой компании (фирмы) также представляет собой не что иное, как совокупность отдельных действий, направленных на достижение требуемого результата. Однако применительно к бизнес-процессам желательно выполнение каждого действия ассоциировать с конкретным подразделением компании. В этом случае подразделение несет ответственность за реализацию отдельных действий, а сам бизнес-процесс представляется в виде переходов действий из одного подразделения к другому.

Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название ***дорожки (swimlanes)***. Имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму. При этом все состояния действия на диаграмме деятельности делятся

на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании (рис. 6).

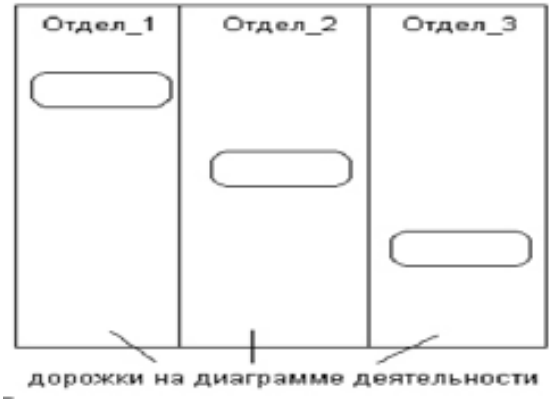


Рисунок 154 – Вариант диаграммы деятельности с дорожками

Названия подразделений явно указываются в верхней части дорожки. Пересекать линию дорожки могут только переходы, которые в этом случае обозначают выход или вход потока управления в соответствующее подразделение компании. Порядок следования дорожек не несет какой-либо семантической информации и определяется соображениями удобства.

В качестве примера рассмотрим фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов по телефону. Подразделениями компании являются отдел приема и оформления заказов, отдел продаж и склад.

Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения. В данном случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое из подразделений торговой компании должно выполнять то или иное действие (рис. 7).

Из указанной диаграммы деятельности сразу видно, что после принятия заказа от клиента отделом приема и оформления заказов осуществляется распараллеливание деятельности на два потока (переход-разделение). Первый из них остается в этом же отделе и связан с получением оплаты от клиента за заказанный товар. Второй инициирует выполнение действия по подбору товара в отделе продаж (модель товара, размеры, цвет, год выпуска и пр.). По окончании этой работы инициируется действие по отпуску товара со склада. Однако подготовка товара к отправке в торговом отделе начинается только после того, как будет получена оплата за товар от клиента и товар будет отпущен со склада (переход-соединение). Только после этого товар отправляется клиенту, переходя в его собственность.

Объекты

В общем случае действия на диаграмме деятельности выполняются над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют некоторый результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности к другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

Для **графического представления объектов** используются прямоугольник класса, с тем отличием, что имя объекта подчеркивается. Далее после имени может указываться характеристика состояния объекта в прямых скобках. Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой. Соответствующая зависимость определяет состояние конкретного объекта после выполнения предшествующего действия.

На диаграмме деятельности с дорожками расположение объекта может иметь некоторый дополнительный смысл. А именно, если объект расположен на границе двух дорожек, то это может означать, что переход к следующему состоянию действия в соседней дорожке ассоциирован с готовностью некоторого документа (объект в некотором состоянии). Если же объект целиком расположен внутри дорожки, то и состояние этого объекта целиком определяется действиями данной дорожки.

Возвращаясь к предыдущему примеру с торговой компанией, можно заметить, что центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до звонка от клиента заказ как объект отсутствует и возникает лишь после такого звонка. Однако этот заказ еще не заполнен до конца, поскольку требуется еще подобрать конкретный товар в отделе продаж. После его подготовки он передается на склад, где вместе с отпуском товара заказ окончательно дооформляется. Наконец, после получения подтверждения об оплате товара эта информация заносится в заказ, и он считается выполненным и закрытым. Данная информация может быть представлена графически в виде модифицированного варианта диаграммы деятельности этой же торговой компании (рис. 8).

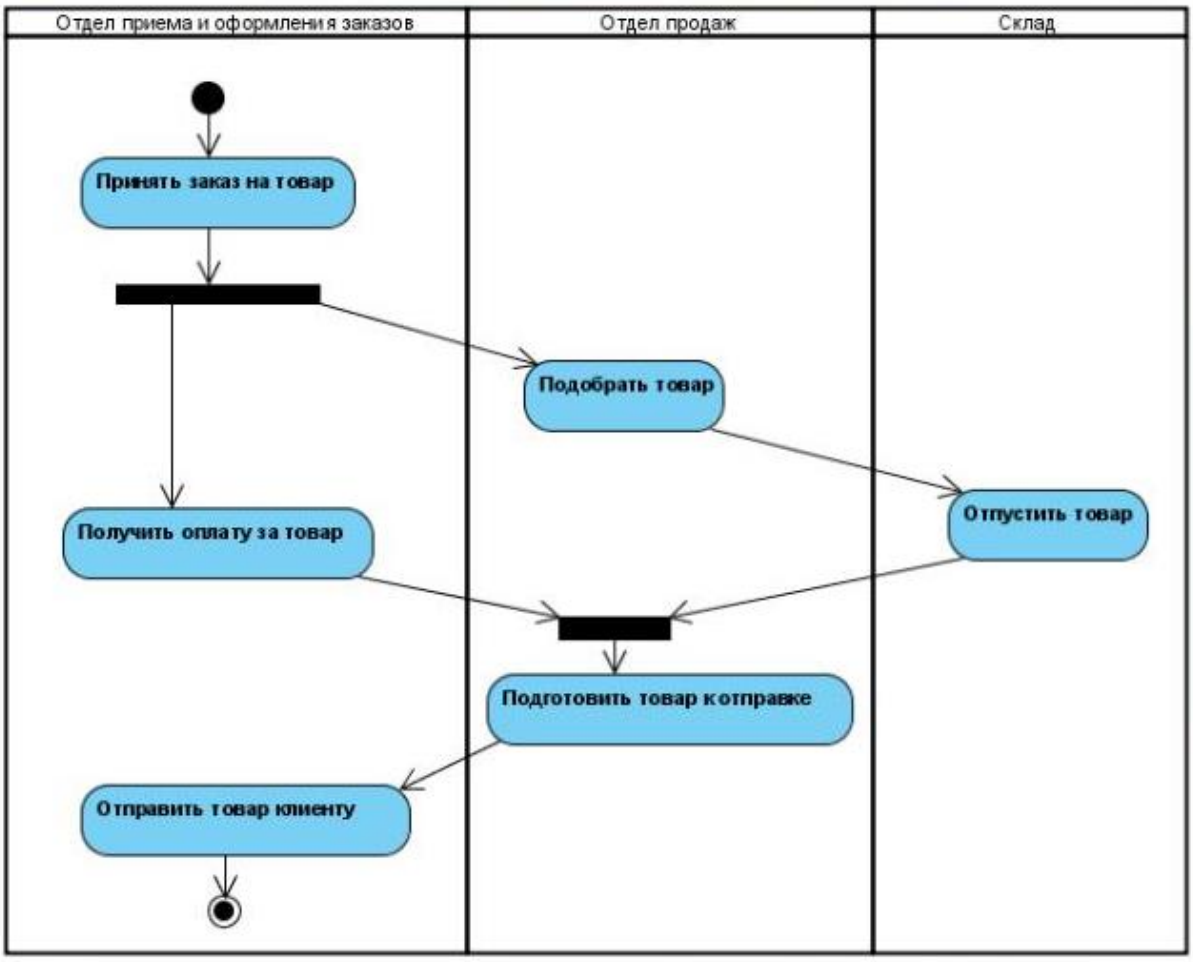


Рисунок 155 – Фрагмент диаграммы деятельности для торговой компании

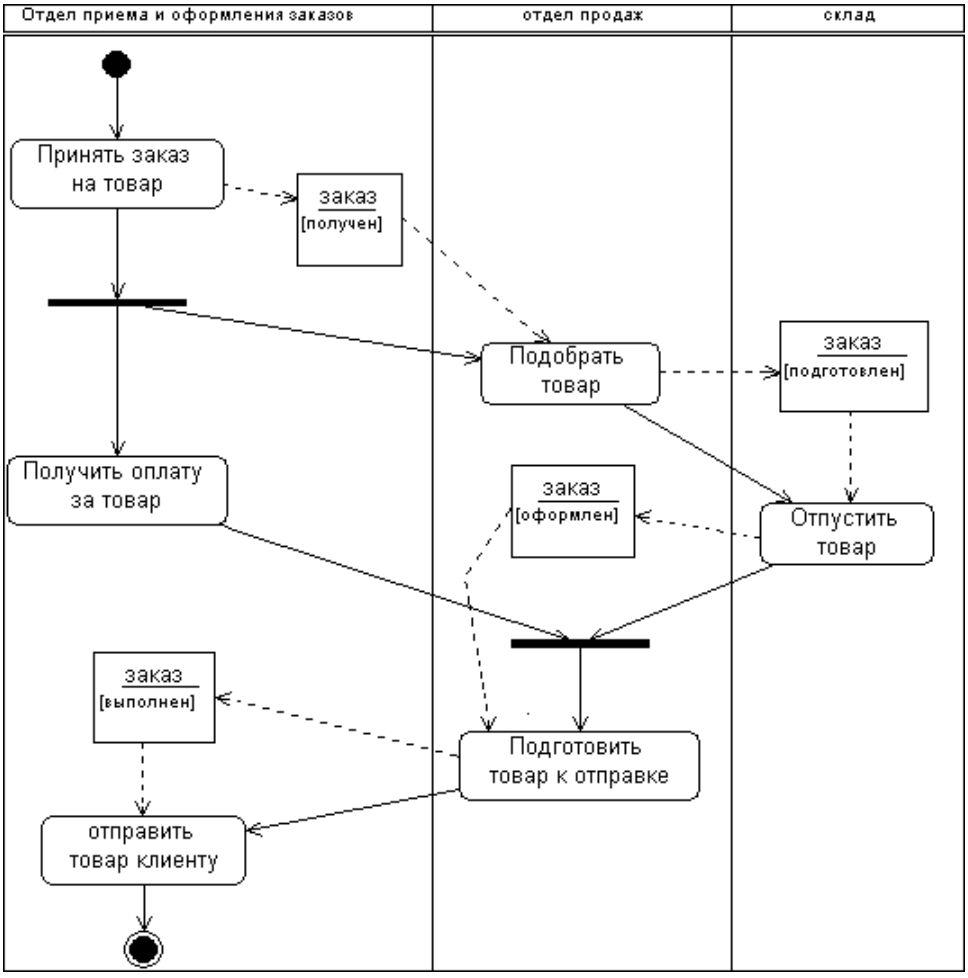


Рисунок 156 – Фрагмент диаграммы деятельности торговой компании с объектом-заказом

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.
2. Откройте файл, созданный в практических занятиях №8-9 и содержащий диаграмму классов.
3. В проводнике по моделям должны отображаться все классы и диаграммы, созданные ранее (рис. 9).

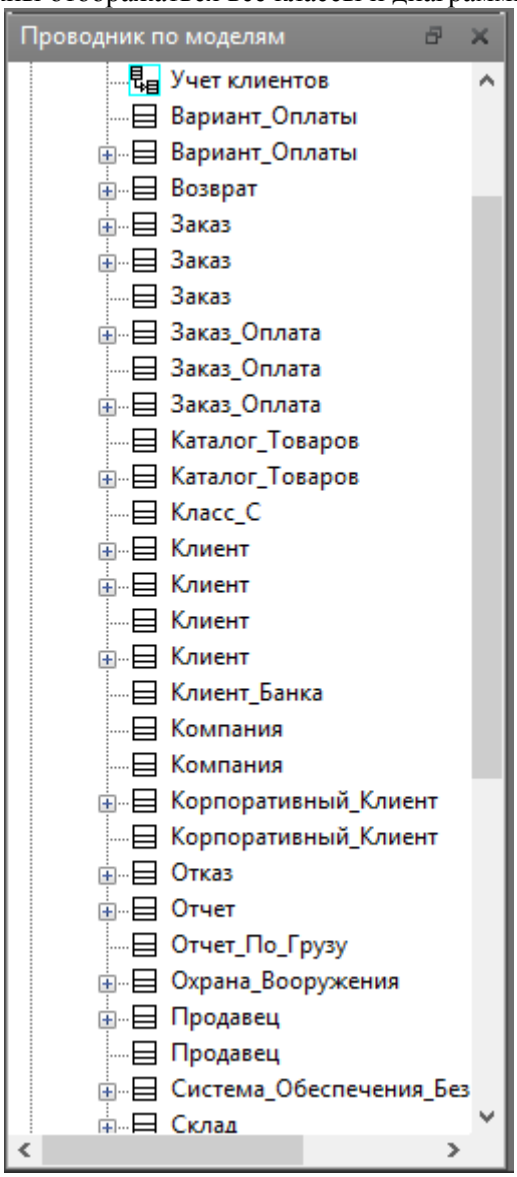


Рисунок 157 – Проводник по моделям

Опишем с помощью диаграммы деятельности процесс формирования заказа и выдачу товара. В бизнес-процессе участвуют 3 действующих лица: клиент, продавец и система оплаты. Следовательно, необходимо добавить 3 дорожки для распределения ответственности между этими лицами.

Для этого, в файле с диаграммой классов, созданной в практическом занятии 8, необходимо проделать следующие действия:

6. Щелкнуть правой кнопкой мыши по *классу Заказ*.
7. В контекстном меню выбрать пункт *Схемы*.
8. Нажать кнопку *Создать* и выбрать *Деятельность*.
9. Переименовать созданный лист в *Деятельность-Заказ*.
10. Построить диаграмму деятельности для класса Заказ. Для это выполните действия, описанные ниже.

a. Добавить 3 элемента *Дорожка* и изменить их названия на *Клиент*, *Продавец* и *Система оплаты* соответственно.

b. Добавить элементы *Начальное состояние*, *Конечное состояние, Состояние действия, Решение, Переход (объединение),* изменить их названия и задать расположение в соответствии с рисунком 10.

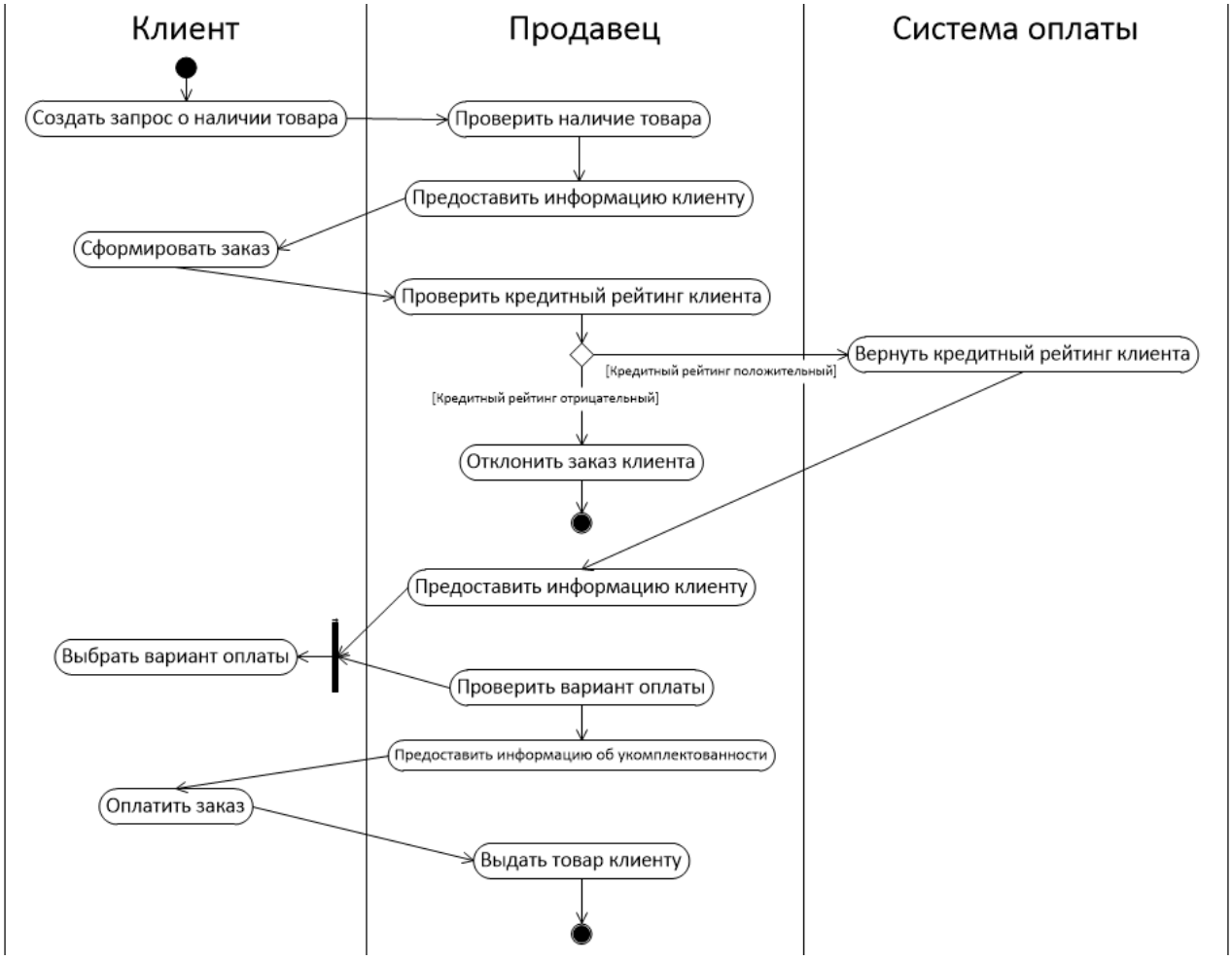


Рисунок 158 – Диаграмма деятельности

5. Задание

Построить диаграмму деятельности в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

Создать диаграммы деятельности не менее чем для трех классов, описанных в практическом занятии №8.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Дайте определение понятию «диаграмма деятельности».
2. Опишите назначение диаграммы деятельности.
3. Дайте определение понятиям «состояние деятельности» и «состояние действия». Графическое изображение состояния.
4. Приведите пример ветвления и параллельных потоков управления процессами на диаграмме деятельности.
5. Какие переходы используются на диаграмме деятельности?
6. Что представляет собой дорожка на диаграмме деятельности?
7. Как графически изображаются объекты на диаграмме деятельности?

Практическая работа №11

Построение диаграмм последовательностей на языке UML с помощью MS Visio

1. Цель работы

Целью работы является изучение основ создания диаграмм последовательностей на языке UML, получение навыков построения диаграмм последовательностей, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

2. Задачи

Основными задачами практической работы являются:

– ознакомиться с теоретическими вопросами построения диаграмм последовательностей на языке UML;

– ознакомиться с теоретическими вопросами построения диаграмм последовательностей с помощью MS Visio.

3. Краткие теоретические сведения

Диаграммы последовательностей описывают взаимодействия множества объектов, включая сообщения, которыми они обмениваются.

В отличие от диаграммы классов, на которой изображаются абстрактные элементы в виде классов, на диаграмме последовательностей используются конкретные экземпляры классов – ***объекты***. Объекты отображаются прямоугольником без полей. Для того чтобы подчеркнуть, что это экземпляр абстрактной сущности, название объекта подчеркивается. При необходимости через двоеточие после названия можно указать сущность (класс) экземпляром которой является этот объект. Отметим, что объект может быть экземпляром не только класса, но и других абстракций, например, актера. Обратите внимание, что при указании в качестве классификатора актера изменится графическое обозначение объекта (рис. 1).

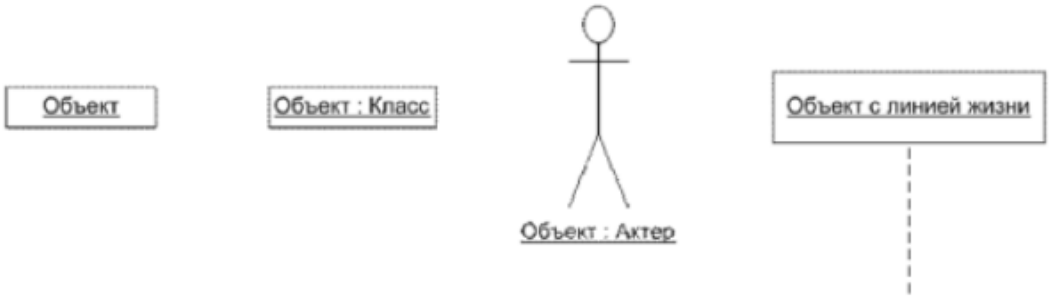


Рисунок 159 – Графическое обозначение объекта UML

На диаграмме последовательностей у объекта может присутствовать ***линия жизни***, на которой отмечаются происходящие с объектом события. Линия жизни отображается пунктирной линией (рис. 2).

Между собой объекты могут быть связаны связями. ***Связь*** – это экземпляр отношения ассоциация, и имеет такое же графическое обозначение, что и ассоциация.

На диаграмме последовательностей объекты обмениваются сообщениями. ***Сообщение*** – это спецификация передачи данных от одного объекта другому, который предполагает какое-то ответное действие. Графически сообщение обозначается сплошной линией со стрелкой.

Часто операция вызывает какую-либо операцию в объекте. Очевидно, что класс, экземпляром которого является объект, должен иметь такую операцию. Привязка сообщения к операции класса объекта выполняется в свойствах сообщения (рис. 2).

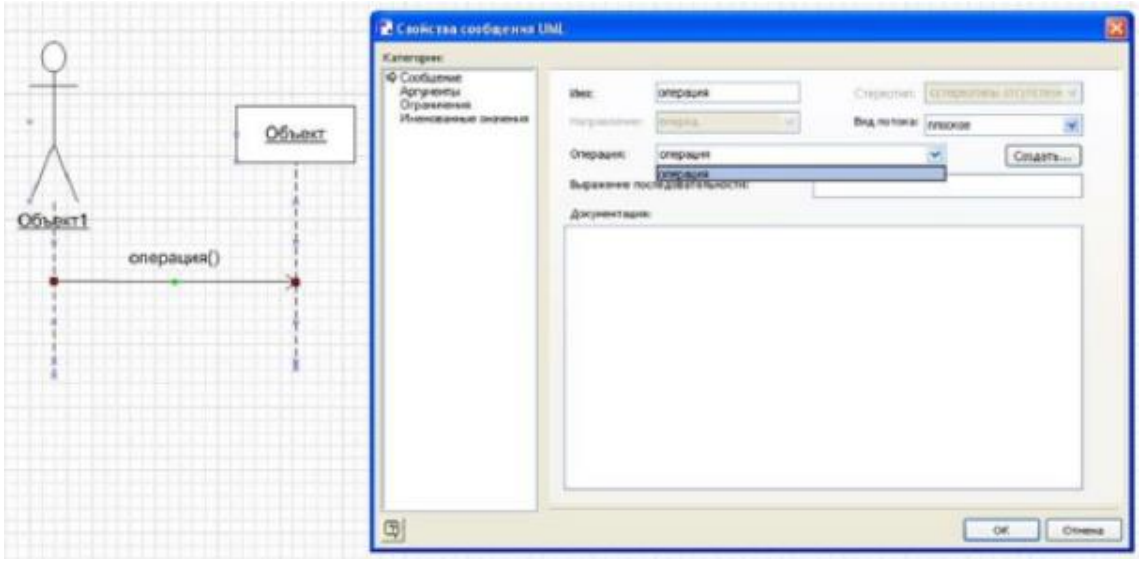


Рисунок 160 – Сообщение UML и его свойства

При построении диаграммы классов обычно определяются только основные свойства сущностей, а такие детали, как операции, удобно создавать при построении диаграммы последовательности, для чего в свойствах сообщения UML есть кнопка создания операции.

Диаграммы последовательностей, как и другие диаграммы для отображения динамических свойств системы, могут быть выполнены в контексте многих сущностей UML. Они могут описывать поведение системы в целом, подсистемы, класса или операции класса и др. К сожалению, Visio недостаточно гибка в плане поддержки раскрытия содержания отдельных элементов с помощью других диаграмм. Например, кликнув правой кнопкой мыши по классу можно обнаружить, что для его описания можно создать лишь диаграммы классов, состояний и деятельности. Поэтому возможность привязать диаграмму последовательностей к элементу, который она реализует, средствами Visio невозможно, эту связь нужно подразумевать.

Диаграммы последовательностей будем делать в контексте прецедентов с диаграммы прецедентов, реализуя те функции, которые должна выполнять наша система.

При построении динамических диаграмм используется уже разработанная структура информационной системы. Для диаграммы последовательностей не нужно придумывать объекты, а достаточно определить, экземпляры каких классов участвуют в этом действии.

Определив необходимые объекты (как экземпляры классов, так и экземпляры актеров), вторым этапом построения диаграмм последовательностей определяются сообщения, пересылаемые между актерами. Фактически определяется последовательность шагов, для выполнения нужного действия.

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.
2. Откройте файл, созданный в практических занятиях №8-10 и содержащий диаграмму классов.
3. В проводнике по моделям должны отображаться все классы и диаграммы, созданные ранее.

Построим диаграмму последовательности для варианта использования «Обеспечить покупателя информацией» (рис. 4). Для этого добавим на диаграмму последовательности линии жизни и соотнесем объекты с актерами, инициирующими вариант использования «Обеспечить покупателя информацией», и с необходимыми классами.

Для **добавления диаграммы последовательности в проект MS Visio** выполните следующие действия:

1. В проводнике по моделям найдите ветку «Основной пакет».
2. Нажмите по ней правой кнопкой мыши > Создать …
3. В контекстном меню выберите пункт «Схема последовательностей».

Добавим сообщения, которыми обмениваются объекты для исполнения варианта использования.

Если объект имеет операцию (посмотреть в практическом занятии №8 «Диаграммы классов» наличие операции у класса, которому принадлежит объект).

1. Из группы фигур «Последовательности UML» добавить три фигуры типа «Линия жизни объекта». Для изменения названия необходимо дважды щелкнуть левой кнопкой мыши по фигуре. Откроется окно свойств объекта и, если в данном файле нет ранее созданных классов, окно создания нового класса. В данном примере

необходимо создать три класса «Товар», «Каталог товаров» и «Заказ» и соответственно три объекта с такими же названиями.

2. С помощью поиска фигур найти фигуру «Актер» и добавить ее в рабочую область. Двойным щелчком левой кнопки мыши задать имя «Клиент».
3. Добавить фигуру «Линия жизни» и соедините ее начало с фигурой «Клиент».
4. Протянуть все линии жизни вниз листа.
5. Добавить фигуры «Сообщение» и соединить, руководствуясь следующими принципами:

5.1. Соединить фигурой «Сообщение» линию жизни клиента с линией жизни объекта товар. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию запросить товар.

5.2. Соединить фигурой «Сообщение» линию жизни клиента с линией жизни объекта заказ. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию сформировать заказ.

5.3. Соединить фигурой «Сообщение» линию жизни объекта товар с линией жизни объекта каталог товаров. Двойным щелчком по сообщению открыть окно свойств и выбрать операцию проверить наличие.

5.4. Соединить фигурой «Сообщение (возврат)» линию жизни объекта товар и линию жизни клиента. Двойным щелчком по сообщению открыть окно свойств и задать текст сообщения «Предоставить информацию».

6. Добавить фигуры «Активация» и расположить их на диаграмме в соответствии с рисунком 4.

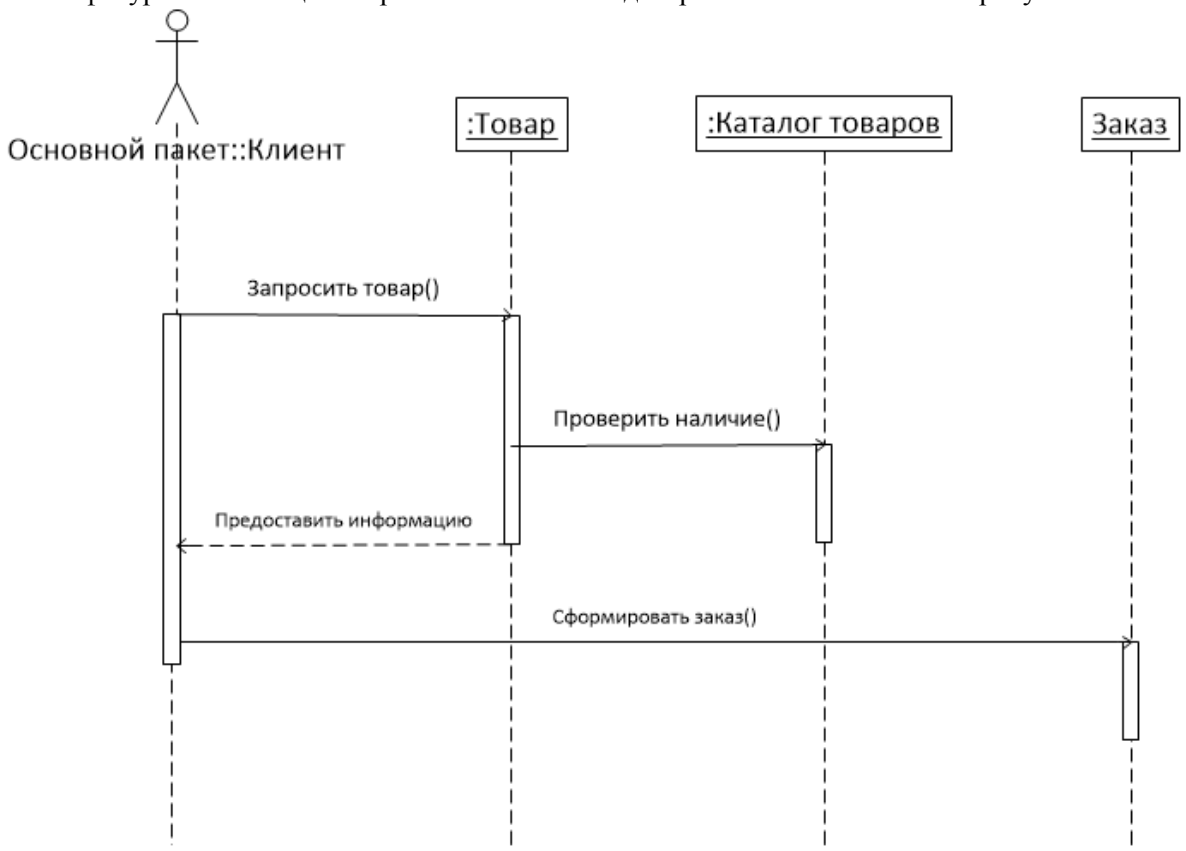


Рисунок 161 – Диаграмма последовательности для варианта использования «Обеспечить покупателя информацией»

При построении диаграмм последовательностей можно вносить коррективы в диаграмму классов. Если объект класса получает новую операцию, то она добавляется в соответствующий класс на диаграмме классов как метод.

Построим диаграмму последовательности для варианта использования «Согласовать условия оплаты» (рис. 5). Действия по построению диаграммы аналогичны построению диаграммы последовательности для варианта использования «Обеспечить покупателя информацией».

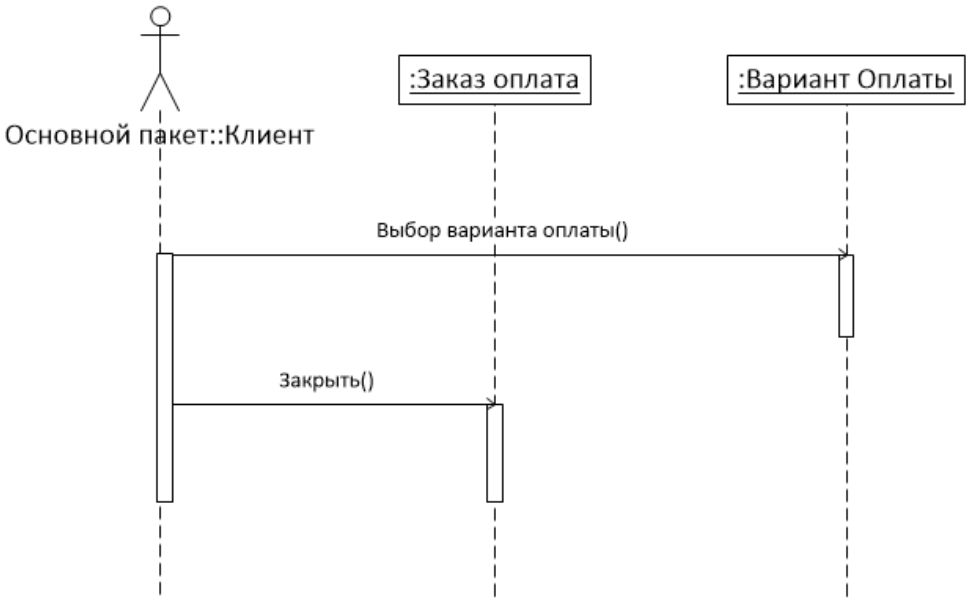


Рисунок 162 – Диаграмма последовательности для варианта использования «Согласовать условия оплаты»

Построим диаграмму последовательности для варианта использования «Заказать товар со склада» (рис. 6). Действия по построению диаграммы аналогичны построению предыдущих диаграмм последовательностей.

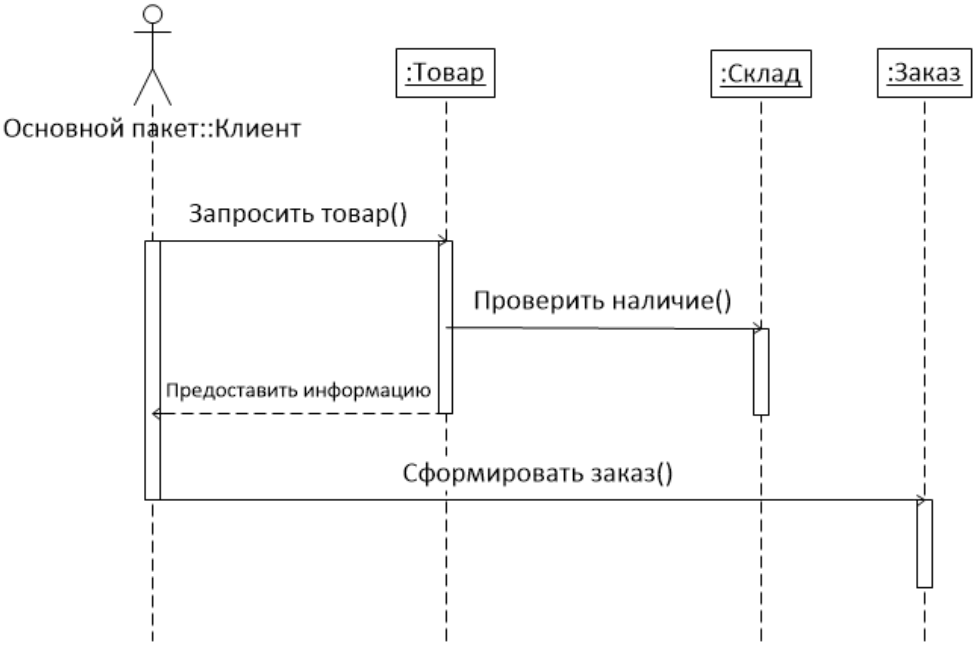


Рисунок 163 – Диаграмма последовательности для варианта использования «Заказать товар со склада»

Построим диаграмму последовательности для системы продажи товаров по каталогу (рис. 7).

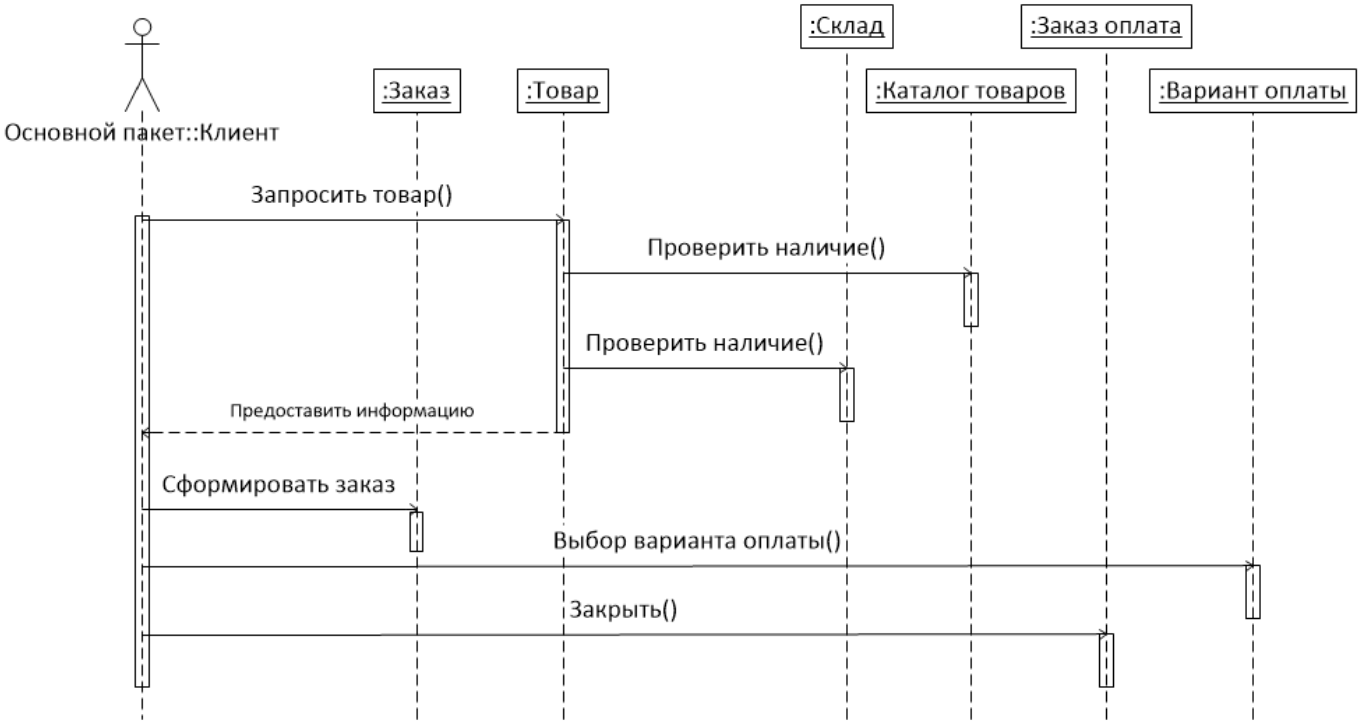


Рисунок 164 – Диаграмма последовательности для системы продажи товаров по каталогу

5. Задание

Построить диаграмму последовательности для каждого варианта использования, определенных в практическом занятии №7 и для всей системы в целом в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Для чего предназначена диаграмма последовательности?
2. Назовите и охарактеризуйте элементы диаграммы последовательности.
3. Что такое сообщение?
4. Что такое линия жизни?
5. Назовите виды сообщений.

Практическая работа №12

Разработка диаграмм прецедентов с помощью Visual Paradigm for UML

1. Цель работы

Целью работы является ознакомление с возможностями case-средства Visual Paradigm for UML Community Edition, освоение основных принципов создания диаграмм вариантов использования с помощью этого case-средства.

2. Задачи

Основными задачами практической работы являются:

– изучение среды case-средства Visual Paradigm for UML Community Edition;
– получить навыки создания диаграмм прецедентов.

3. Краткие теоретические сведения

3.1. Интерфейс Visual Paradigm for UML CE

Visual Paradigm for UML Community Edition – среда объектно-ориентированного проектирования на языке UML, распространяемая бесплатно для некоммерческого использования.

При первых запусках среда будет предлагать сообщить своё имя и e-mail для получения регистрационного кода. Получив код по электронной почте, следует активировать лицензию. Некоторые функции среды доступны только в её платных версиях. Начальное окно среды Visual Paradigm for UML Community Edition имеет вид, изображенный на рисунке 1.

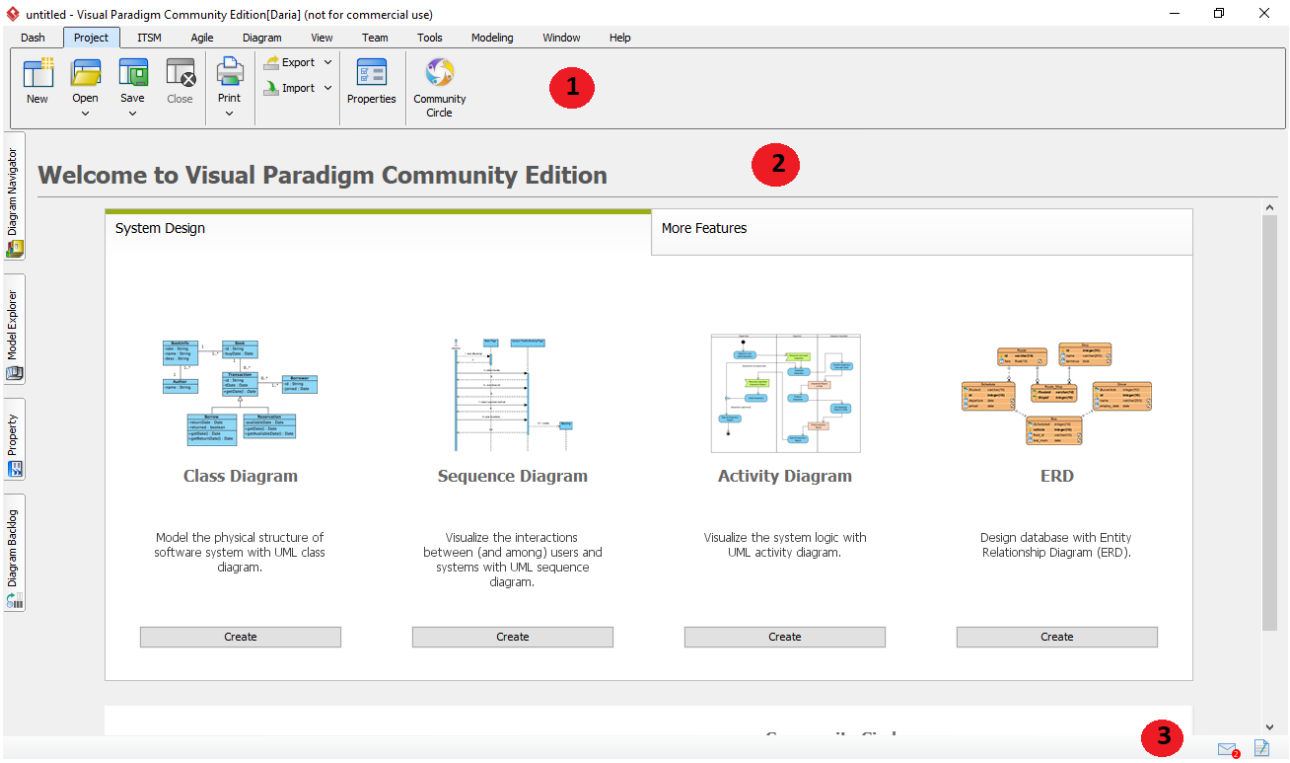


Рисунок 165 – Начально окно Visual Paradigm for UML CE

В таблице 1 приведено описание основных элементов интерфейса среды объектно-ориентированного проектирования Visual Paradigm for UML Community Edition.

Таблица 11 – Описание элементов интерфейса Visual Paradigm for UML CE

№ п/п	Наименование	Описание
1	Панель инструментов	Панель со вкладками, позволяющая выполнять основные операции по созданию проектов и работе с диаграммами в Visual Paradigm
2	Редактор диаграмм	Рабочая область, внутри которой отображается проектируемая диаграмма
3	Строка состояния	Строка, в которой отображаются уведомления

Сохранение и открытие проектов

Для сохранения созданного проекта необходимо выбрать в панели инструментов *Project* > *Save* или *Project* > *Save as*… При первом сохранении проекта, среда проектирования предложит выбрать место для сохранения.

Для открытия ранее созданного проекта необходимо выбрать в панели инструментов *Project* > *Open* и указать пусть к файлу проекта, имеющему расширение *Visual Paradigm Project (*.vpp)*.

Основы работы с диаграммами в Visual Paradigm for UML CE

Далее описываются основные шаги по созданию диаграмм, добавлению фигур и установлению связей между ними.

1. Создание диаграмм

Рассмотрим в качестве примера процесс создания диаграммы вариантов использования. Для создания диаграммы прецедентов необходимо выполнить следующую последовательность действий.

1) Выбрать *Diagram* > *New* а панели инструментов (рис. 2).
2) Начать набор названия диаграммы в строке поиске, в данном случае *use case diagram.*

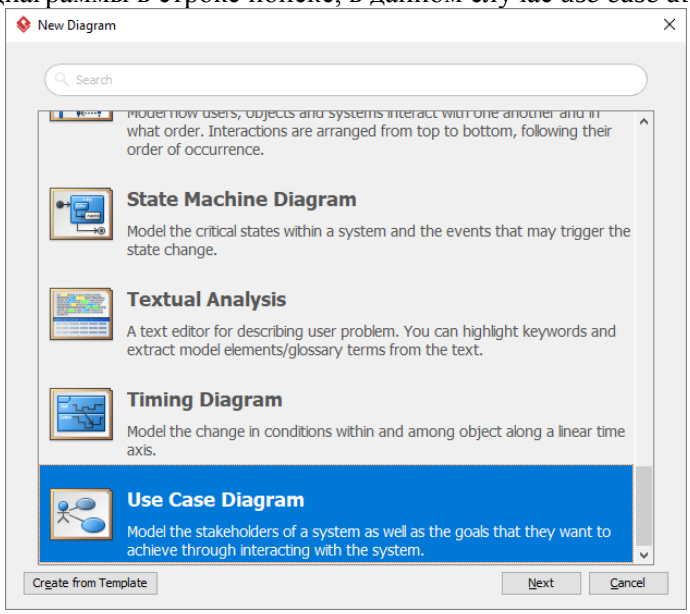


Рисунок 166 – Окно создания новой диаграммы

3) Выбрать диаграмму в списке и нажать кнопку *Next.*
4) Задать имя диаграммы, место расположения и описание.
5) Нажать кнопку *Ok.*

2. Создание и соединение фигур

2.1. Используя панель инструментов для работы с диаграммами

Далее рассмотрим процесс добавления фигуры *Актер* с помощью панели инструментов для работы с диаграммами (рис. 3).

1) Нажать на кнопку с изображением *Актера* в панели инструментов для работы с диаграммами.
2) Щелкнуть в рабочей области диаграммы чтобы создать актера и ввести его название.
3) Нажать в произвольном месте рабочей области либо нажать клавишу *Enter* на клавиатуре.

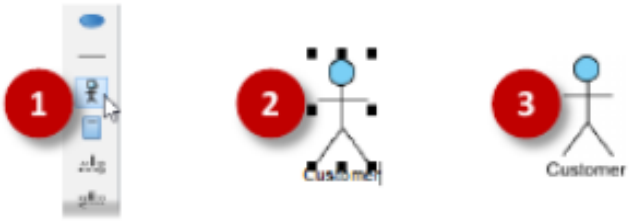


Рисунок 167 – Добавление фигуры Актер

2.2. Используя каталог ресурсов

Переместив курсор мыши на фигуру в ее правом верхнем углу появляется значок, который предназначен для доступа к каталогу ресурсов.

Каталог ресурсов позволяет создавать новую фигуру, которая автоматически соединяется с существующей. Также его можно использовать для создания связи между двумя существующими фигурами.

Рассмотрим последовательность действий для создания варианта использования от существующего актера (рис. 4).

1) Переместить курсор мыши на фигуру *Актер*.
2) Нажать на кнопку *Каталог ресурсов* в правом верхнем углу и потянуть указатель вправо.
3) Отпустить кнопку мыши и выбрать *Association > Use Case*.
4) Ввести название варианта использования.

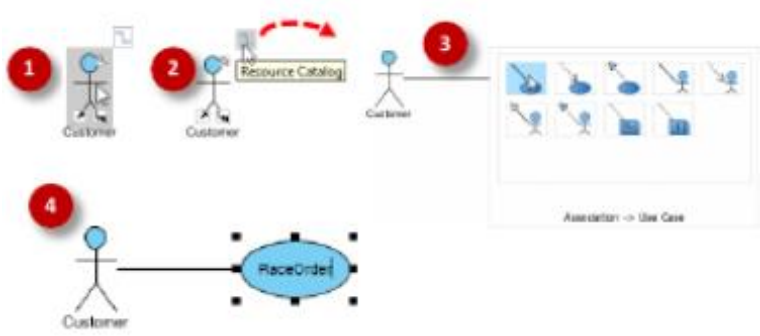


Рисунок 168 – Добавление варианта использования с помощью каталога ресурсов

3. Изменение размера фигур

После нажатия на фигуру появляется несколько обработчиков изменения размера. Перетаскивая их можно изменять размер фигуры.

4. Добавление контрольных точек к соединениям

Для большинства диаграмм в качестве соединения по умолчанию используется «наклонное соединение», представляющее собой прямую линию. Добавление контрольных точек позволит изменить положение линии (рис. 5).

Для добавления контрольной точки необходимо навести курсор мыши на соединение нажать левую кнопку мыши и не отпуская ее переместить точку в нужное место.

Рисунок 169 – Добавление контрольной точки

Примечание. Существует 5 типов соединений. Чтобы применить другой тип необходимо щелкнуть правой кнопкой мыши по линии и выбрать *Styles and Formatting > Connector Styles* и выбрать подходящий тип. Если необходимо изменить сразу все соединения нужно щелкнуть правой кнопкой мыши в свободном месте рабочей области и выбрать *Connectors* во всплывающем меню.

5. Изменение цвета фигуры

Для придания более выразительного внешнего вида диаграмме можно изменять цвет фигур.

Рассмотрим последовательность действий для изменения цвета варианта использования.

1) Щелкнуть правой кнопкой мыши по фигуре *Вариант использования* и в контекстном меню выбрать *Styles and Formatting > Formats…*
2) Открыть вкладку *Backgrounds* в окне *Formats.* Выбрать нужный цвет и нажать кнопку *Ok* чтобы применить выбранный цвет к фигуре.

Экспорт диаграммы

Готовую диаграмму можно полностью или частично копировать в формате изображения JDG или EMF в буфер обмена для экспорта в другое приложение.

Для этого следует выделить нужные объекты или всю диаграмму (Ctrl+A) и в контекстном меню выбрать команду *Copy > Copy to Clipboard as Image (JPG)*. Данная команда дублируется комбинацией клавиш Ctrl+Alt+C. После выполненных действий изображение диаграммы готово к вставке в другом приложении.

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

4. Запустите visual Paradigm for UML CE.
5. Создайте новый проект: *Project* > *New*. Введите название проекта и нажмите кнопку *Create Blank Project* (рис. 6).

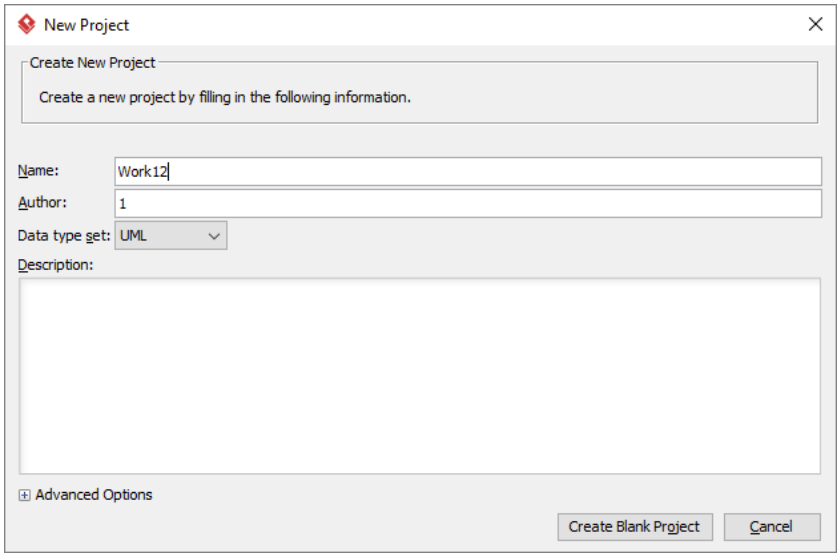


Рисунок 170 – Окно создания проекта

6. Создайте новую диаграмму вариантов использования (*Use Case Diagram*): *Diagram* > *New* (рис. 7).

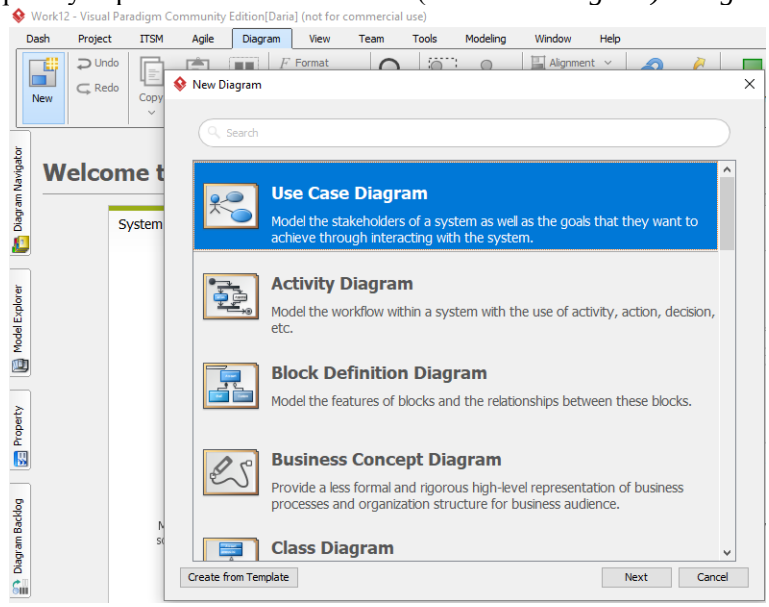


Рисунок 171 – Добавление диаграммы в проект

7. Присвойте имя диаграмме: *SaleOfGoods* (рис. 8).

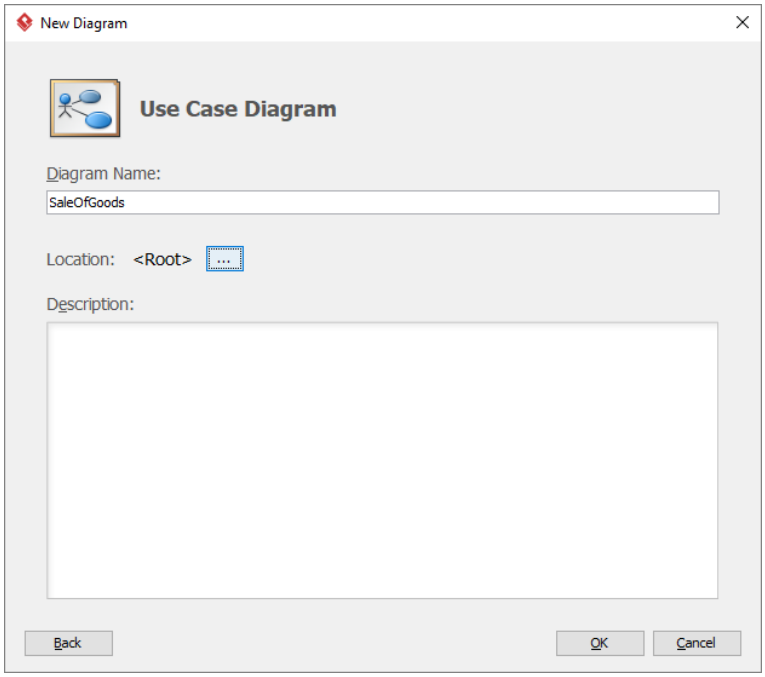


Рисунок 172 – Указание имени диаграммы

8. Добавьте фигуру *System*. Для этого в панели инструментов для работы с фигурами выберите нужную фигуру и щелкните в любом месте рабочей области. После этого переименуйте систему задав название: «Система продажи товаров по каталогу» (рис. 9).

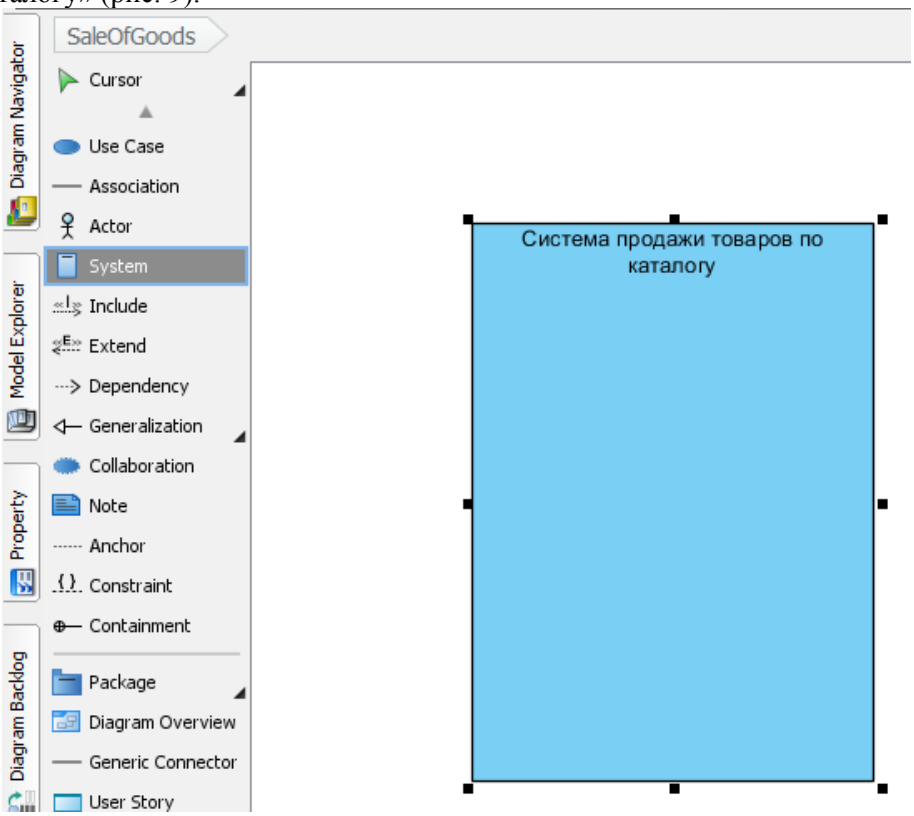


Рисунок 173 – Добавление элемента Система

9. Далее добавьте актеров *Продавец* и *Покупатель* и вариант использования *Оформить заказ на покупку товара.*

10. Добавьте связь актеров с вариантом использования. Для этого: выделите актера > в правом верхнем углу откройте каталог ресурсов > в контекстном меню выберите use case > во вспомогательном окне начните вводить название ранее созданного прецедента «Оформить заказ на покупку товара» > нажмите кнопку Ok.

Результат выполнения действий 6 и 7 показан на рисунке 10.

Рисунок 174 – Исходная диаграмма прецедентов

11.Далее необходимо установить кратность для отношения ассоциации между актерами и вариантом использования. Для этого необходимо навести курсор на связь и дважды щелкнуть левой кнопкой мыши по точке вначале линии. В появившемся контекстном окне необходимо задать кратность данной стороны отношения (рис. 11).

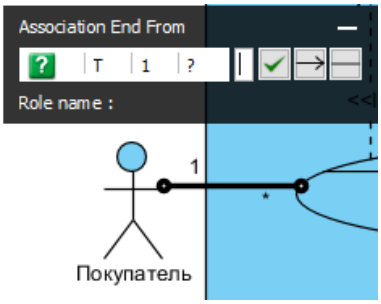


Рисунок 175 – Задание кратности отношения

Для задания кратности второй стороне отношения нажмите кнопку со стрелкой вправо и проделайте те же действия. Для того, чтобы применить произведенные действия достаточно нажать кнопку с зеленой галочкой либо щелкнуть в любом пустом месте рабочей области.

12.Дополним исходную диаграмму. Для этого добавим 4 варианта использования «Обеспечить покупателя информацией», «Согласовать условия оплаты», «Заказать товар со склада» и «Запросить каталог товаров» (рис. 12).

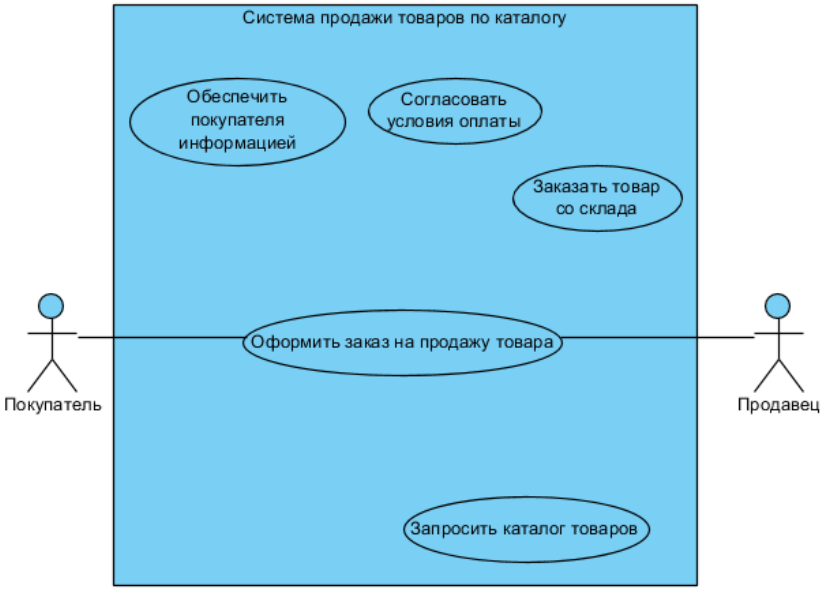


Рисунок 176 – Дополнительные варианты использования

13.Установим связи между основным и дополнительными вариантами использования. «Запросить каталог товаров» связан с «Оформить заказ на продажу товара» отношением расширения. Для того чтобы установить этот тип соединения необходимо в панели инструментов для работы с диаграммами выбрать соответствующую фигуру (рис. 13).

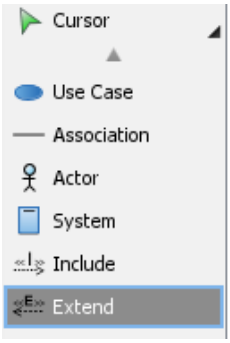


Рисунок 177 – Фигура отношения расширения

После чего соединить два варианта использования.

«Обеспечить покупателя информацией», «Согласовать условия оплаты», «Заказать товар со склада» связаны с «Оформить заказ на покупку товара» отношением включения. Для того чтобы установить этот тип соединения необходимо в панели инструментов для работы с диаграммами выбрать соответствующую фигуру (рис. 13).

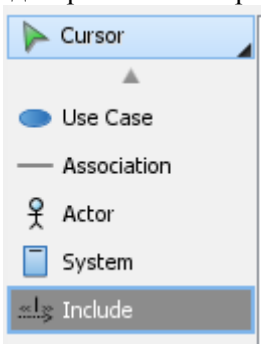


Рисунок 178 – Фигура отношения включения

После чего соединить варианты использования «Обеспечить покупателя информацией», «Согласовать условия оплаты», «Заказать товар со склада» с «Оформить заказ на покупку товара».

В результате выполнения всех вышеописанных действий диаграмма вариантов использования должна иметь вид, показанный на рисунке 14.

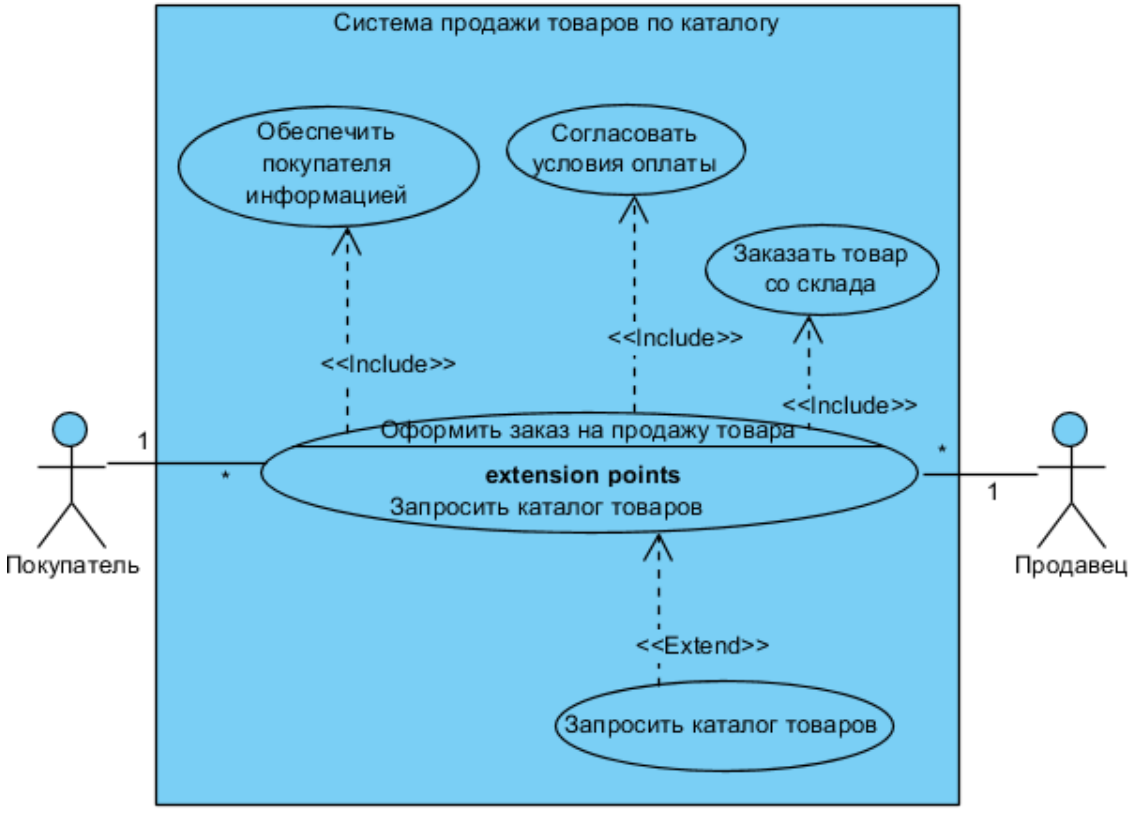


Рисунок 179 – Дополненная диаграмма вариантов использования

5. Задание

Построить диаграмму прецедентов (вариантов использования) в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения организационной диаграммы, а также скриншоты результатов согласно заданию.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Для чего используется диаграмма вариантов использования (прецедентов) Use Case?
2. Назовите основные элементы диаграммы прецедентов.
3. Как создать диаграмму вариантов использования с помощью Visual Paradigm for UML CE?хм

Практическая работа №13

Разработка диаграмм классов с помощью Visual Paradigm for UML

1. Цель работы

Целью работы является изучение основ создания диаграмм классов на языке UML с помощью case-средства Visual Paradigm for UML CE, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

2. Задачи

Основными задачами практической работы являются:

– получить навыки создания диаграмм классов с помощью case-средства Visual Paradigm for UML CE.

3. Краткие теоретические сведения

Диаграмма классов отражает различные взаимосвязи между отдельными сущностями предметной области, а также описывает их внутреннюю структуру и типы отношений.

Диаграмма классов (class diagram) – диаграмма языка UML, на которой представлена совокупность декларативных или статических элементов модели, таких как классы с атрибутами и операциями, а также связывающие их отношения.

Диаграмма классов предназначена для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования.

Диаграмма классов содержит интерфейсы, пакеты, отношения и даже отдельные экземпляры классификаторов, такие как объекты и связи.

Класс (class) – абстрактное описание множества однородных объектов, имеющих одинаковые атрибуты, операции и отношения с объектами других классов.

Графически изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 1). В этих секциях могут указываться имя класса, атрибуты и операции класса. Даже если секции атрибутов и операций пусты, в обозначении класса они должны быть выделены горизонтальной линией.

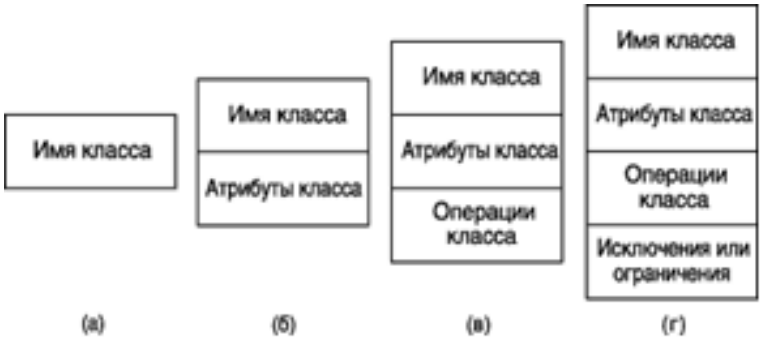


Рисунок 180 – Варианты графического изображения класса на диаграмме классов

В первом случае для класса Окружность (рис. 2, а) указаны только его атрибуты – точка на координатной плоскости, которая определяет расположение ее центра. Для класса Окно (рис. 2, б) указаны только его операции, при этом секция его атрибутов оставлена пустой. Для класса Счет (рис. 2, в) дополнительно изображена четвертая секция, в которой указано требование – реализовать резервное копирование объектов этого класса.

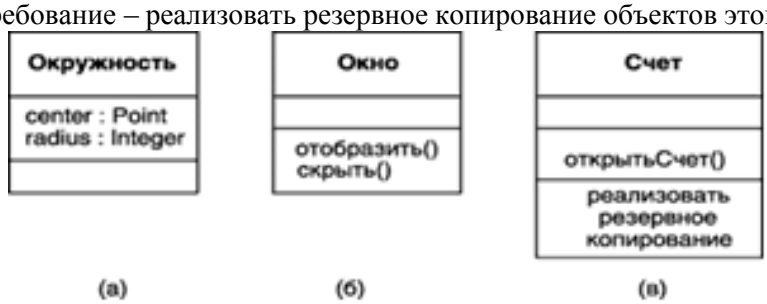


Рисунок 181 – Примеры графического изображения конкретных классов

Имя класса должно быть уникальным в пределах пакета, который может содержать одну или несколько диаграмм классов. Имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы.

В секции имени класса могут также находиться стереотипы или ссылки на стандартные шаблоны, от которых образован данный класс и, соответственно, от которых он наследует атрибуты и операции. Здесь также могут записываться и другие общие свойства этого класса.

Класс может иметь или не иметь экземпляров или объектов. В зависимости от этого в **языке UML различают** конкретные и абстрактные классы.

Конкретный класс (concrete class) – класс, на основе которого могут быть непосредственно созданы экземпляры или объекты.

Рассмотренные выше обозначения относятся к конкретным классам.

Абстрактный класс (abstract class) – класс, который не имеет экземпляров или объектов.

Для обозначения имени абстрактного класса используется курсив.

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

Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения.

Общий формат записи отдельного атрибута класса следующий:

<квантор видимости> <имя атрибута> [кратность] : <тип атрибута> = <исходное значение> {строка-свойство}

Видимость (visibility) – качественная характеристика описания элементов класса, характеризующая потенциальную возможность других объектов модели оказывать влияние на отдельные аспекты поведения данного класса.

Видимость в языке UML специфицируется с помощью *квантора видимости* (может быть опущен), который может принимать одно из 4-х возможных значений и отображаться при помощи специальных символов.

– "+" – область видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.

– "#" – область видимости типа защищенный (protected). Атрибут недоступен или невиден для всех классов, за исключением подклассов данного класса.

– "-" – область видимости типа закрытый (private). Атрибут недоступен или невиден для всех классов без исключения.

– "~" – область видимости типа пакетный (package). Атрибут недоступен или невиден для всех классов за пределами пакета, в котором определен класс-владелец данного атрибута.

Вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private, package.

Имя атрибута используется в качестве идентификатора соответствующего атрибута и поэтому должно быть уникальным в пределах данного класса. Имя атрибута - обязательный элемент, должно начинаться со строчной (малой) буквы и не должно содержать пробелов.

Кратность (multiplicity) – спецификация области значений допустимой мощности, которой могут обладать соответствующие множества.

Тип атрибута представляет собой выражение, семантика которого определяется некоторым типом данных, определенным в пакете Типы данных языка UML или самим разработчиком, или в зависимости от языка программирования, который предполагается использовать для реализации данной модели.

Исходное значение служит для задания начального значения соответствующего атрибута в момент создания отдельного экземпляра класса. Если оно не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса.

Пояснительный текст в фигурных скобках может означать две различные конструкции. Если в этой строке имеется знак равенства, то вся запись Строка-свойство 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.

Знак "/" перед именем атрибута указывает на то, что данный атрибут является производным от некоторого другого атрибута этого же класса.

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.

Общий формат записи отдельной операции класса следующий:

<квантор видимости> <имя операции>(список параметров):

<выражение типа возвращаемого значения> {строка-свойство}

Квантор видимости операции аналогичен квантору видимости атрибутов.

Имя операции – обязательный элемент, должно начинаться со строчной (малой) буквы, и записываться без пробелов.

Список параметров является перечнем разделенных запятой формальных параметров, каждый из которых, в свою очередь, может быть представлен в следующем виде:

<направление параметра> <имя параметра> : <выражение типа> = <значение параметра по умолчанию>

Параметр (parameter) – спецификация переменной операции, которая может быть изменена, передана или возвращена.

Выражение типа возвращаемого значения указывает на тип данных значения, которое возвращается объектом после выполнения соответствующей операции. Две точки и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения. Для указания нескольких возвращаемых значений данный элемент спецификации операции может быть записан в виде списка отдельных выражений.

Кроме внутреннего устройства классов важную роль при разработке проектируемой системы имеют различные отношения между классами, которые также могут быть изображены на диаграмме классов. Базовые отношения, изображаемые на диаграммах классов:

– отношение ассоциации (association relationship);
– отношение обобщения (generalization relationship);
– отношение агрегации (aggregation relationship);
– отношение композиции (composition relationship).

Отношение ассоциации соответствует наличию произвольного отношения или взаимосвязи между классами. Оно обозначается сплошной линией со стрелкой или без нее и с дополнительными символами, которые характеризуют специальные свойства ассоциации.

В качестве простого примера ***ненаправленной бинарной ассоциации*** можно рассмотреть отношение между двумя классами – классом «Компания» и классом «Сотрудник» (рис. 3). Они связаны между собой бинарной ассоциацией «Работает». Для данного отношения определен следующий порядок чтения следования классов - сотрудник работает в компании.

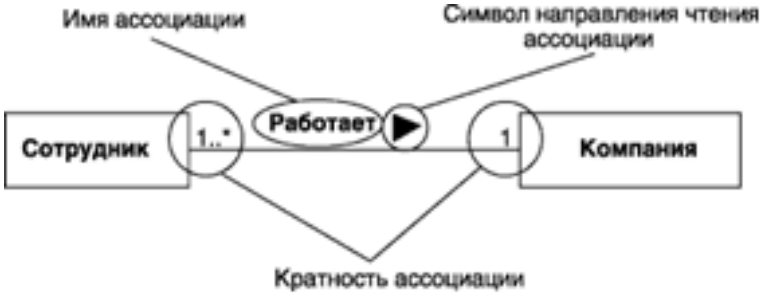


Рисунок 182 – Графическое изображение ненаправленной бинарной ассоциации между классами

Направленная бинарная ассоциация изображается сплошной линией с простой стрелкой на одной из ее концевых точек. Направление этой стрелки указывает на то, какой класс является первым, а какой – вторым.

В качестве простого примера направленной бинарной ассоциации можно рассмотреть отношение между двумя классами – классом «Клиент» и классом «Счет» (рис. 4). Они связаны между собой бинарной ассоциацией с именем «Имеет», для которой определен порядок следования классов. Это означает, что конкретный объект класса «Клиент» всегда должен указываться первым при рассмотрении взаимосвязи с объектом класса «Счет», например, <клиент, счет_1, счет_2,…, счет_n>.

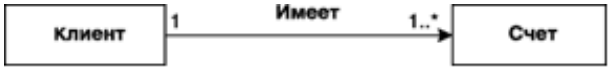


Рисунок 183 – Графическое изображение направленной бинарной ассоциации между классами

Частный случай отношения ассоциации – так называемая *исключающая_ассоциация* (Xor-association). Семантика данной ассоциации указывает на то, что из нескольких потенциально возможных вариантов данной ассоциации в каждый момент времени может использоваться только один.

На диаграмме классов исключающая ассоциация изображается пунктирной линией, соединяющей две и более ассоциации (рис. 5), рядом с которой записывается ограничение в форме строки текста в фигурных скобках: {xor}.

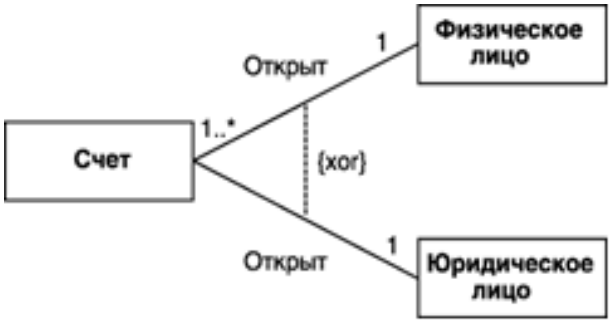


Рисунок 184 – Графическое изображение исключающей ассоциации между тремя классами

Отношение обобщения является отношением классификации между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком).

Менее общий элемент модели должен быть согласован с более общим элементом и может содержать дополнительную информацию. Отношение обобщения описывает иерархическое строение классов и наследование их свойств и поведения.

Согласно одному из главных принципов методологии ООП – наследованию, класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет собственные свойства и поведение, которые могут отсутствовать у класса-предка.

Отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рис. 6). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее начало – на более специальный класс (класс-потомок или подкласс).

Рисунок 185 – Графическое изображение отношения обобщения в языке UML

От одного класса-предка одновременно могут наследовать несколько классов-потомков. т.е. в класс-предок входит несколько линий со стрелками.

Пример: класс «Транспортное средство» (курсив обозначает абстрактный класс) может выступать в качестве суперкласса для подклассов, соответствующих конкретным транспортным средствам, таким как: «Автомобиль», «Автобус», «Трактор» и другим (рис 7).

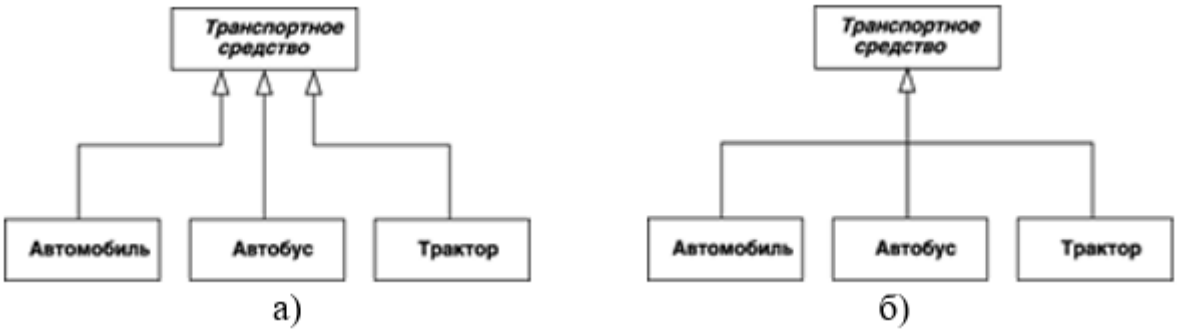


Рисунок 186 – а) Пример графического изображения отношения обобщения для нескольких классов-потомков; б) Альтернативный вариант графического изображения отношения обобщения классов для случая объединения отдельных линий

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой сущность, которая включает в себя в качестве составных частей другие сущности. Данное отношение применяется для представления системных взаимосвязей типа «часть-целое» и показывает, из каких элементов состоит система, и как они связаны между собой.

Очевидно, что рассматриваемое в таком аспекте деление системы на составные части представляет собой иерархию, но принципиально отличную от той, которая порождается отношением обобщения. Отличие заключается в том, что части системы никак не обязаны наследовать ее свойства и поведение, поскольку являются самостоятельными сущностями. Более того, части целого обладают собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого. Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой не закрашенный внутри ромб. Этот ромб указывает на тот класс, который представляет собой «целое» или класс-контейнер. Остальные классы являются его «частями» (рис. 8).

Рисунок 187 – Графическое изображение отношения агрегации в языке UML

В качестве примера отношения агрегации можно рассмотреть взаимосвязь типа «часть-целое», которая имеет место между классом «Системный блок» персонального компьютера и его составными частями: «Процессор», «Материнская плата», «Оперативная память», «Жесткий диск» и «Дисковод гибких дисков» (рис. 9).

Рисунок 188 – Диаграмма классов для иллюстрации отношения агрегации на примере системного блока ПК

Отношение композиции – частный случай отношения агрегации, и служит для спецификации более сильной формы отношения «часть-целое», при которой составляющие части тесно взаимосвязаны с целым. Особенность: с уничтожением целого уничтожаются и все его составные части.

Пример (рис. 10): Программное средство представляет собой базу данных «Автоматизация процесса составления расписания в учебном заведении». Программное средство обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствие с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

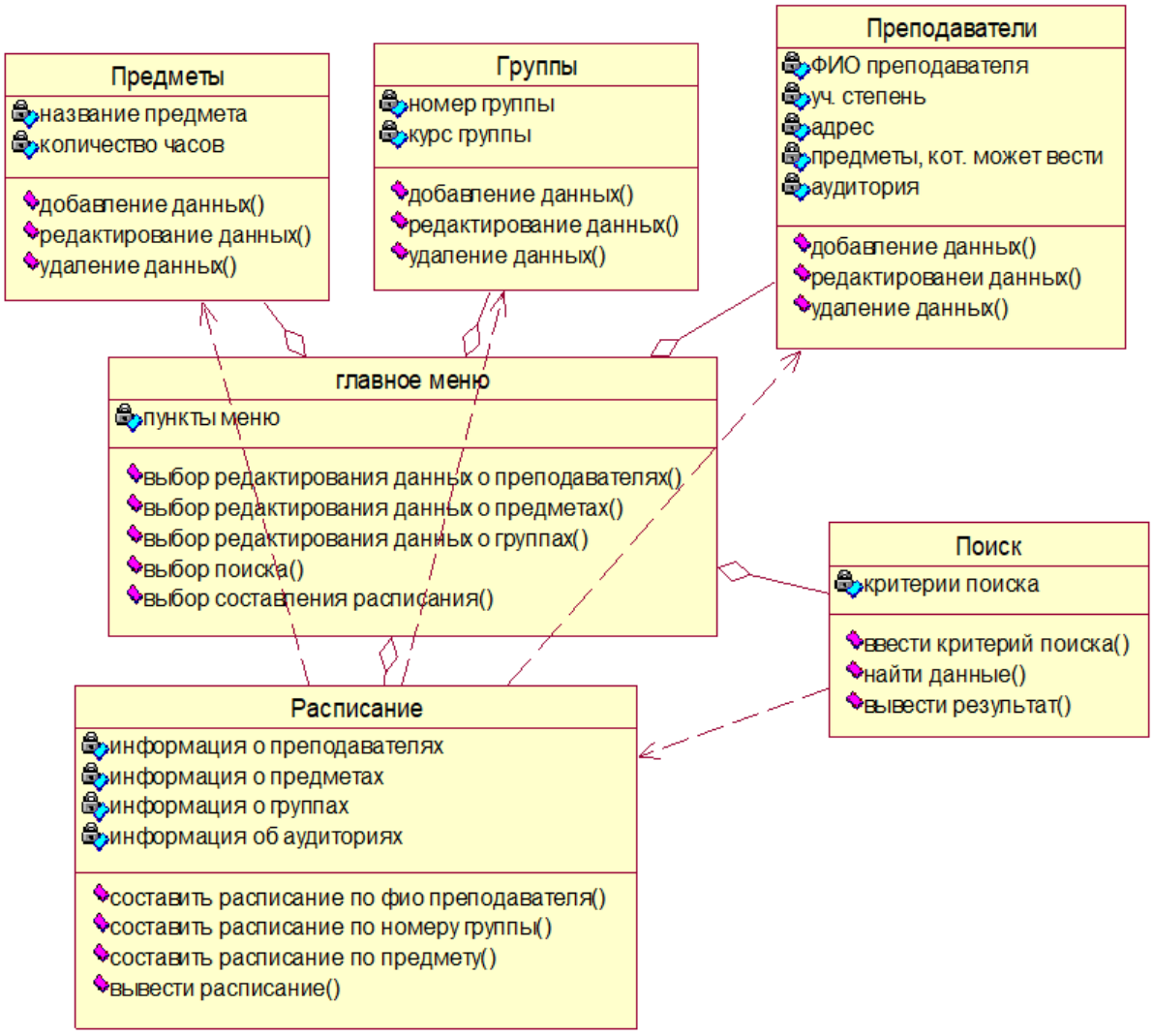


Рисунок 189 – Пример диаграммы классов

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите Visual Paradigm for UML CE.
2. Откройте проект, содержащий диаграмму вариантов использования, построенную в практическом занятии №12.
3. Для добавления новой диаграммы достаточно в меню Diagram > New Diagram выбрать тип диаграммы Class Diagram и нажмите кнопку Next.
4. Задайте имя диаграммы SaleSystemClassDiagram. Нажмите OK.
5. В результате отобразится пустая рабочая область с элементами для построения диаграммы классов (рис. 11).

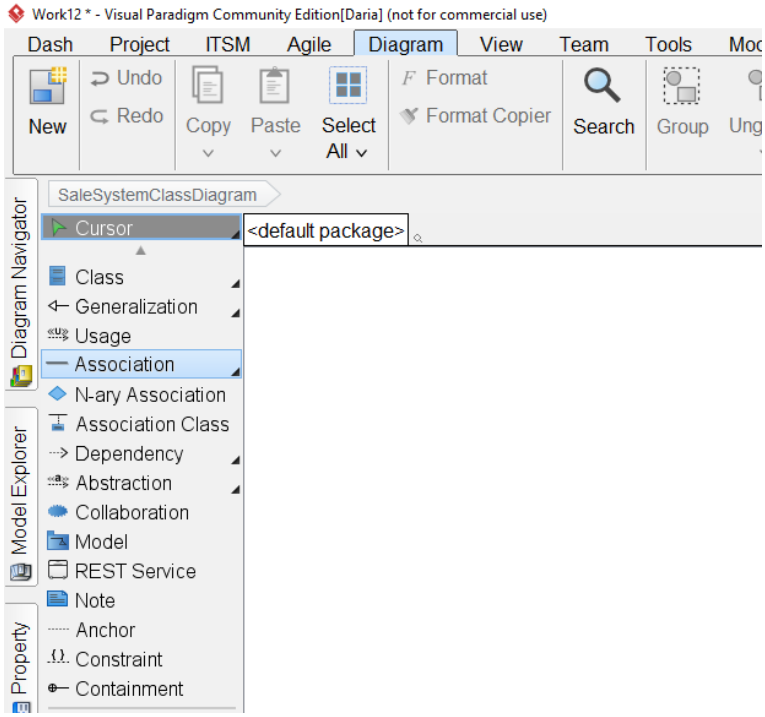


Рисунок 190 – Рабочая область для построения диаграммы классов

Далее будут описаны принципы построения диаграммы классов.

Для создания класса необходима на панели инструментов выбрать Class и затем курсором мыши выбрать место расположения на диаграмме (Рисунок 12). После создания класса необходимо задать его имя (Рисунок 13).

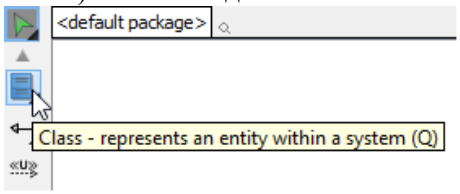


Рисунок 191 – Создание класса

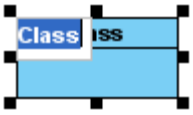


Рисунок 192 – Задание имени в созданном классе

Для создания ассоциативной связи между классами выберите **Association** > **Class (**Рисунок 14**).**

Перетащите курсор мыши на пустое место на диаграмме для создания нового или на уже существующий класс для соединения. Отпустите кнопку мыши (Рисунок 15)**.**

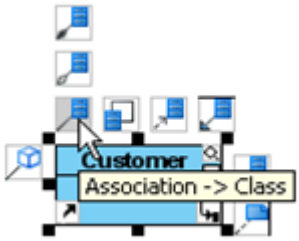


Рисунок 193 – Выбор типа связи

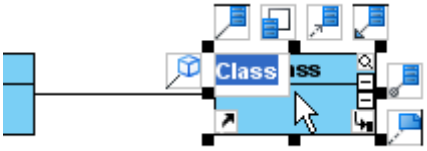


Рисунок 194 – Создание ассоциативной связи

Для создания связи агрегации между классами выберите **Aggregation** -> **Class (**Рисунок 19**).**

Рисунок 195 – Создание связи агрегации

Чтобы изменить кратность ассоциативной связи, щелкните правой кнопкой мыши на конце ассоциации, выберите **Multiplicity** из всплывающего меню, а затем выберите кратность (Рисунок 17). Либо выделите связи и нажмите **Enter** (рисунок 18).

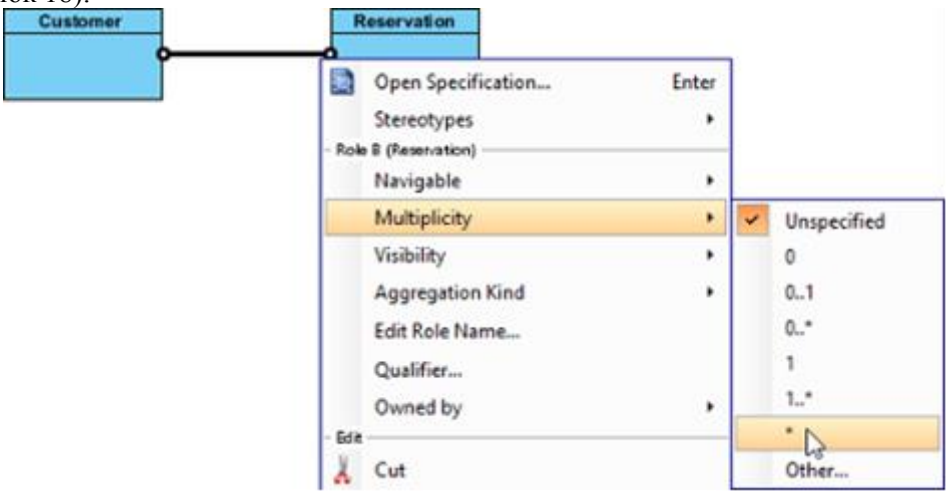


Рисунок 196 – Изменение кратности связи

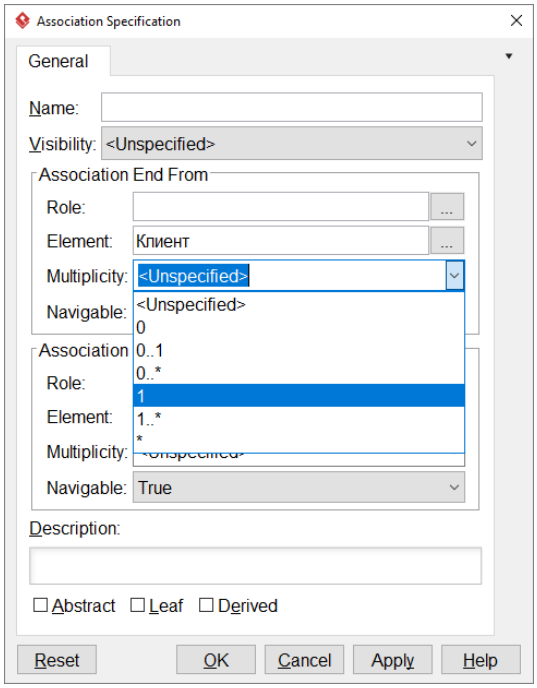


Рисунок 197 – Изменение кратности связи

Чтобы показать направление ассоциации, щелкните правой кнопкой мыши на связи и выберите **Presentation Options** > **Show Direction** из всплывающего меню (Рисунок 19).

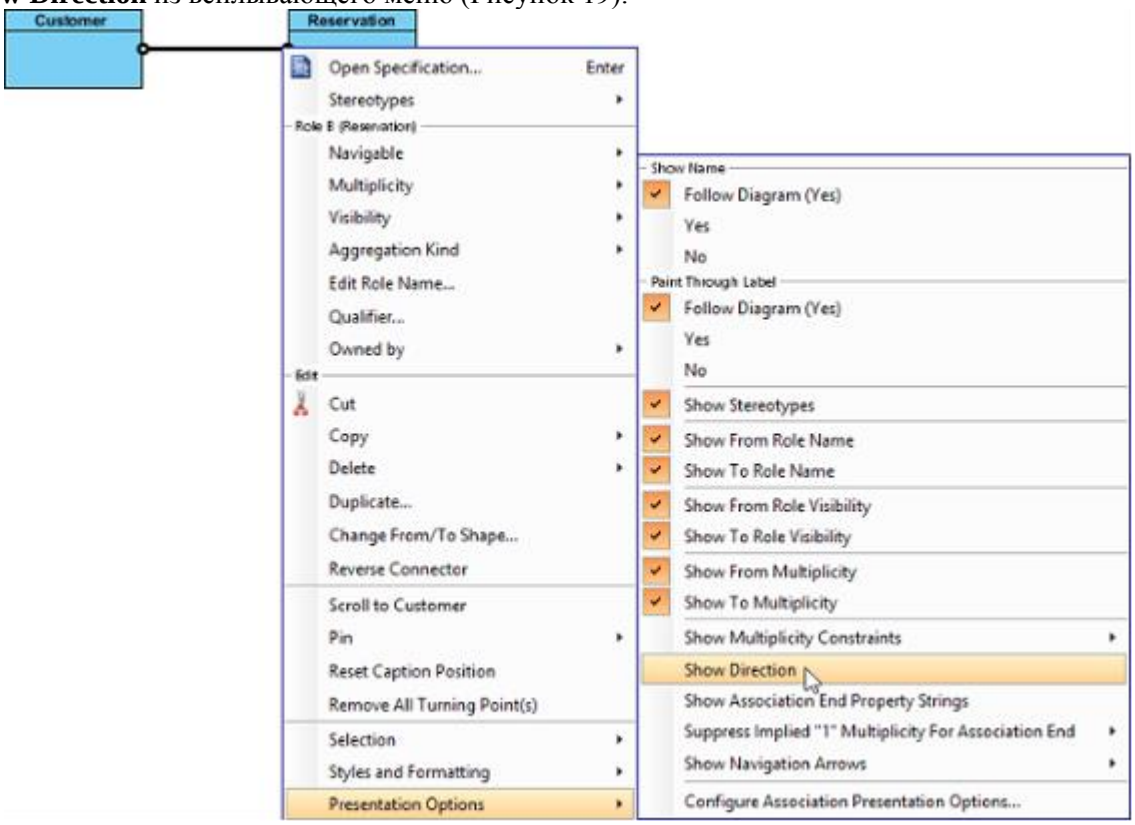


Рисунок 198 – Определение направления связи

Направление стрелки покажет связанный элемент диаграммы (Рисунок 20).

Рисунок 199 – Результат определения ассоциации

Для создания связи обобщения между классами выберите **Generalization -> Class (**Рисунок 23**).**

Перетащите курсор мыши на пустое место на диаграмме для создания нового или на уже существующий класс для соединения. Отпустите кнопку мыши (Рисунок 24)**.**

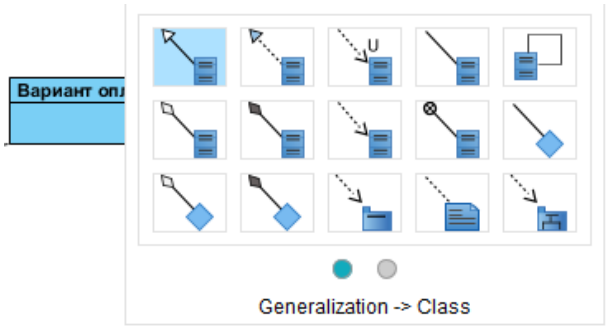


Рисунок 200 – Выбор связи

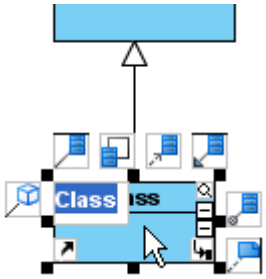


Рисунок 201 – Создание связи обобщения

Для создания атрибута щелкните правой кнопкой мыши на классе и выберете **Add** > **Attribute** в выпадающем меню (Рисунки 23-24).

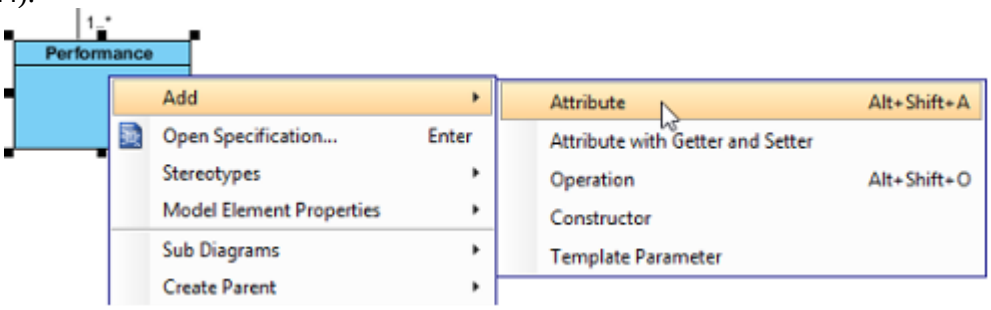


Рисунок 202 – Создание атрибута

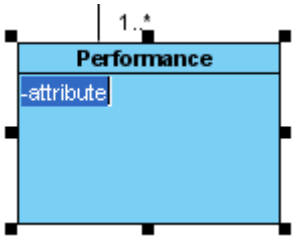


Рисунок 203 – Класс с созданным атрибутом

Для ускорения процесса создания атрибутов нужно сразу же после созданного атрибута на жать на клавиатуре клавишу **Enter**.

Для добавления типа данных атрибута выделите его и нажмите **Enter** либо щелкните правой кнопкой мыши и выберите **Open Specification…** в окне свойств выберите тип данных (рис. 25).

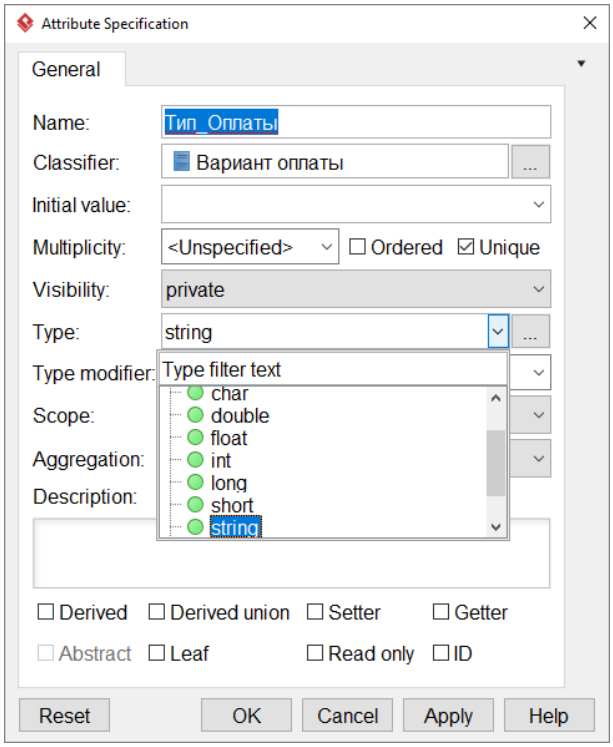


Рисунок 204 – Задание типа данных атрибута

Для создания операции щелкните правой кнопкой мыши на классе и выберете **Add** > **Operation** в выпадающем меню (Рисунки 26-27).

Рисунок 205 – Создание операции

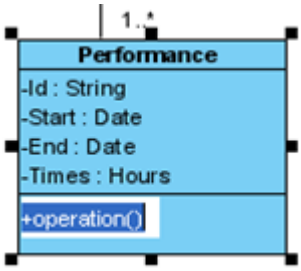


Рисунок 206 – Созданная операция

Для того чтобы переупорядочить члены класса необходимо зацепить курсором мышки один из членов и перетащить (Рисунки 28-29)

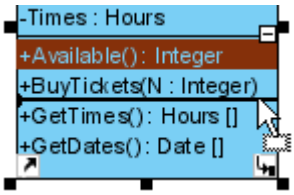


Рисунок 207 – Перетаскивание члена класса

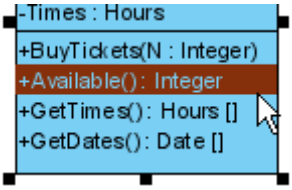


Рисунок 208 – Результат перетаскивания

Для копирования члена из одного класса в другой необходимо выбрать курсором мыши этот член класс с зажатой клавишей **Ctrl** и указать другой класс (Рисунки 30-31).

Рисунок 209 – Копирование члена класса

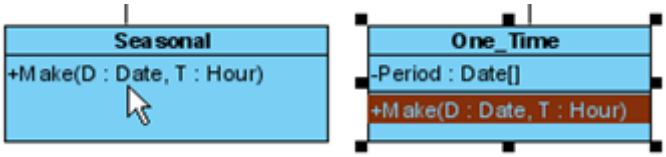


Рисунок 210 – Результат копирования члена класса

Перетаскивание члена класса из одного класса в другой производится подобным образом что и копирование, за исключением того, что не нужно зажимать клавишу **Ctrl**.

Для того чтобы выбрать все члены класса необходимо выбрать один член класса и нажать сочетание клавиш **Ctrl+A**.

Связывание классов производится по следующему алгоритму.

1. Выбрать тип связи из панели инструментов (Рисунок 32).

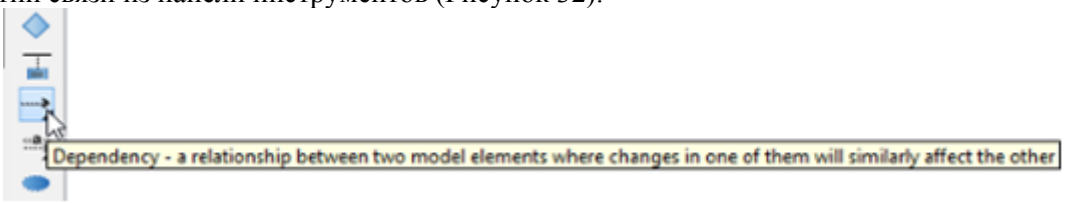


Рисунок 211 – Выбор зависимости

2. Переместить курсор мыши на атрибут-источник исходного класса (Рисунок 33).

Рисунок 212 – Определение связи по члену класса

3. Зажать клавишу мыши и не отпускать ее.
4. Переместить курсор мыши на член другого класса для связи (Рисунок 34).

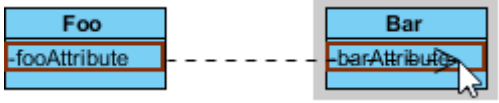


Рисунок 213 – Определение члена класса в целевом классе

5. Отпустить клавишу мыши для определения соединения. В спецификации можно убедиться, что связь определена именно по указанным атрибутам (Рисунок 35).

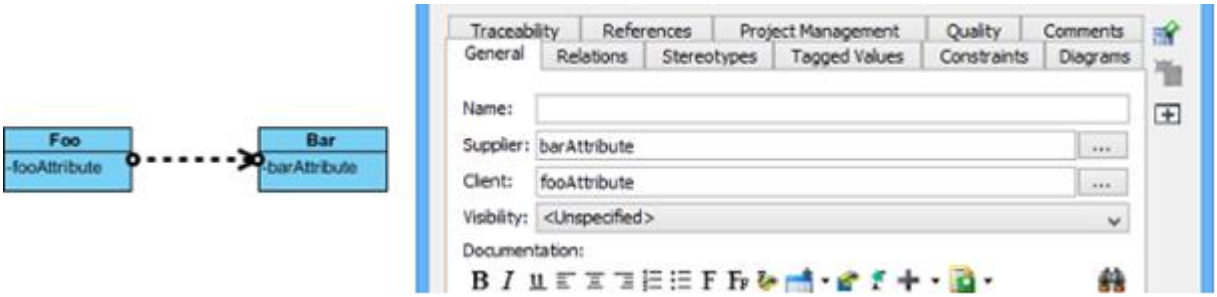


Рисунок 214 – Созданная связь

Основываясь, на описанных выше принципах построения диаграммы классов, постройте диаграмму классов для системы продажи товаров по каталогу в соответствии с рисунком 36.

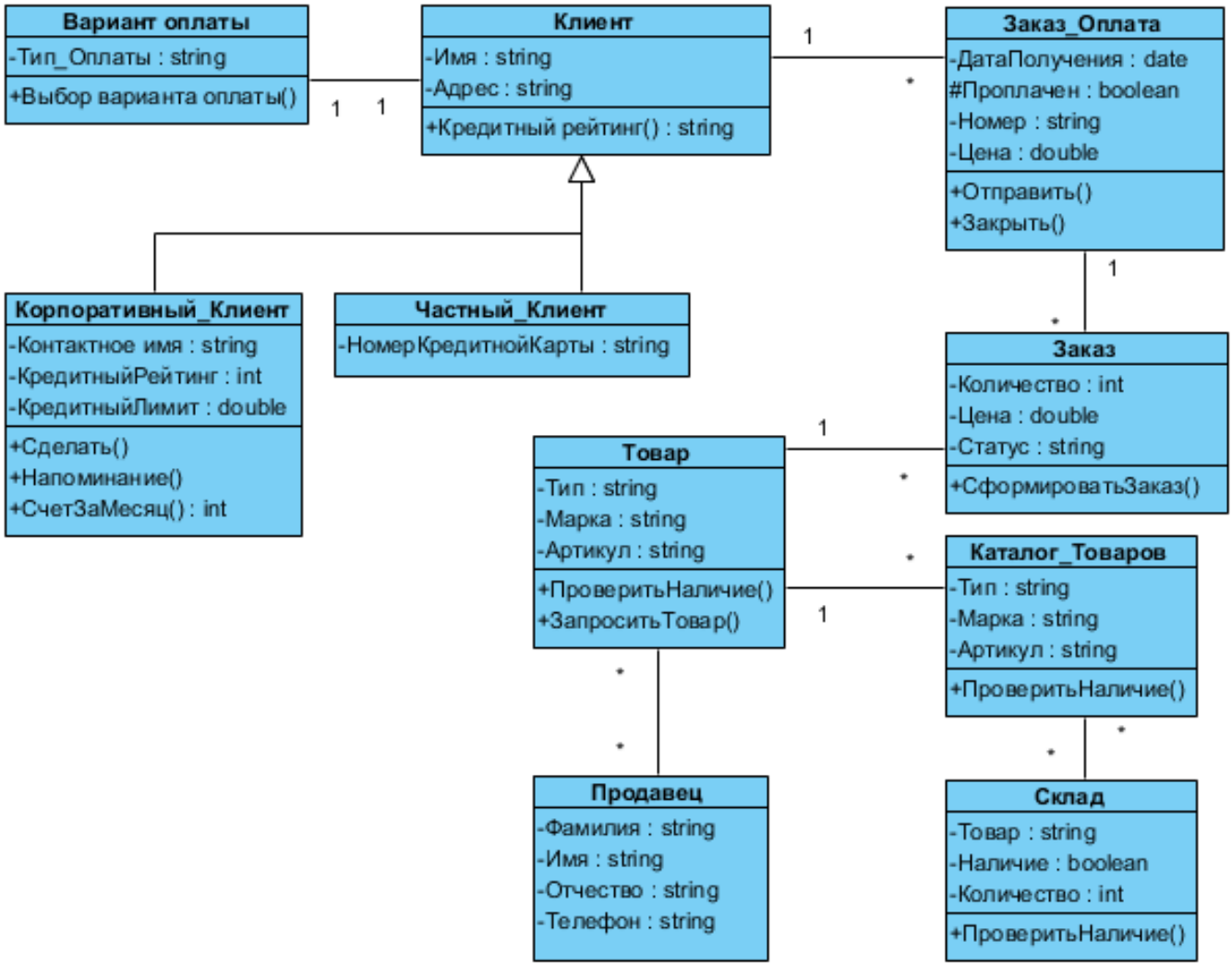


Рисунок 215 – Фрагмент диаграммы классов, описывающей реализацию систем продаж товаров по каталогу

5. Задание

Построить диаграмму классов в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;

7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Дайте определение понятию «диаграмма классов» и ее назначение.
2. Дайте определение следующим понятиям «класс», «атрибут», «операция», «отношение».
3. Опишите отношение ассоциаций между экземплярами классов.
4. Опишите отношение обобщения между экземплярами классов.
5. Опишите отношение агрегации между экземплярами классов.
6. Опишите отношение композиции между экземплярами классов.
7. Приведите пример графического представления основных компонентов диаграммы классов.
8. Дайте определение «квантор видимости» и его классификацию.

Практическая работа №14

Разработка диаграмм состояний с помощью Visual Paradigm for UML

1. Цель работы

Целью работы является изучение основ создания диаграмм состояний на языке UML с помощью case-средства Visual Paradigm for UML CE, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

2. Задачи

Основными задачами практической работы являются:

– получить навыки создания диаграмм состояний с помощью case-средства Visual Paradigm for UML CE;
– изучить основные стереотипы классов.

3. Краткие теоретические сведения

Для представления динамических особенностей взаимодействия элементов модели, в контексте реализации вариантов использования, предназначены диаграммы кооперации и последовательности. Однако для моделирования процессов функционирования большинства сложных систем, особенно систем реального времени, этих представлений недостаточно.

Диаграмма состояний (statechart diagram) – диаграмма, которая представляет конечный автомат, в которой вершины обозначают состояния, а дуги показывают переходы между двумя состояниями.

Назначение диаграммы состояний – описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы в течение всего ее жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий.

Диаграммы состояний используются для описания поведения отдельных систем и подсистем. Они также могут быть применены для спецификации функциональности экземпляров отдельных классов, т. е. для моделирования всех возможных изменений состояний конкретных объектов. Диаграмма состояний по существу является графом специального вида, который служит для представления конечного автомата.

Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы для более детального представления состояний отдельных элементов модели. Для понимания семантики конкретной диаграммы состояний необходимо представлять особенности поведения моделируемой сущности, а также иметь общие сведения из теории конечных автоматов.

Конечный автомат (state machine) – модель для спецификации поведения объекта в форме последовательности его состояний, которые описывают реакцию объекта на внешние события, выполнение объектом действий, а также изменение его отдельных свойств.

В контексте языка UML понятие конечного автомата обладает дополнительной семантикой. Вершинами графа конечного автомата являются состояния и другие типы элементов модели, которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Конечный автомат описывает поведение отдельного объекта в форме последовательности состояний, охватывающих все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет собой конечный автомат.

Состояние (state) – условие или ситуация в ходе жизненного цикла объекта, в течение которого он удовлетворяет логическому условию, выполняет определенную деятельность или ожидает события.

Состояние может быть задано в виде набора конкретных значений атрибутов объекта некоторого класса, при этом изменение отдельных значений этих атрибутов будет отражать изменение состояния моделируемого объекта или системы в целом. Однако не каждый атрибут класса может характеризовать состояние его объектов. Как правило, имеют значение только те свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения. В этом случае состояние будет характеризоваться некоторым инвариантным условием, включающим в себя только принципиальные для поведения объекта или системы атрибуты классов и их значения.

Такое условие может соответствовать ситуации, когда моделируемый объект находится в состоянии ожидания возникновения внешнего события. В то же время нахождение объекта в некотором состоянии может быть связано с выполнением определенных действий. В последнем случае соответствующая деятельность начинается в момент перехода моделируемого элемента в рассматриваемое состояние, а после и элемент может покинуть данное состояние в момент завершения этой деятельности.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рис. 1). Этот прямоугольник может быть разделен на две секции. Если указана лишь одна секция, то в ней записывается только имя состояния (рис. 1, а). В противном случае в первой из них записывается имя состояния, а во второй – список некоторых внутренних действий или переходов в данном состоянии (рис. 1, б). При этом под действием в языке

UML понимают некоторую атомарную операцию, выполнение которой приводит к изменению состояния или возврату некоторого значения (например, «истина» или «ложь»).

Рисунок 216 – Графическое изображение состояний на диаграмме состояний

Имя у состояния может отсутствовать, т. е. оно необязательно для некоторых состояний. В этом случае состояние является *анонимным*. Если на одной диаграмме состояний несколько анонимных состояний, то все они должны различаться между собой.

Действие (action) – спецификация выполнимого утверждения, которая образует абстракцию вычислительной процедуры.

Каждое действие записывается в виде отдельной строки и имеет следующий формат:

<метка действия '/ ' выражение действия>

Метка действия указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия. При этом выражение действия может использовать любые атрибуты и связи, принадлежащие области имен или контексту моделируемого объекта. Если список выражений действия пустой, то метка действия с разделителем в виде наклонной черты '/' не указывается. Перечень меток действий в языке UML фиксирован, причем эти метки не могут быть использованы в качестве имен событий:

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. 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.

Пример: аутентификацию клиента для доступа к ресурсам моделируемой информационной системы (рис. 2). Список внутренних действий в данном состоянии может включать следующие действия. Первое действие – входное, которое выполняется при входе в это состояние и связано с получением строки символов, соответствующих паролю клиента. Далее выполняется деятельность по проверке введенного клиентом пароля. При успешном завершении этой проверки выполняется действие на выходе, которое отображает меню доступных для клиента опций.

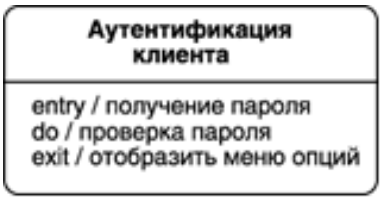


Рисунок 217 – Пример состояния с непустой секцией внутренних действий

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 (рис. 3, а), из которого может только выходить стрелка-переход.

Рисунок 218 – Графическое изображение начального и конечного состояний

На самом верхнем уровне представления объекта переход из начального состояния может быть помечен событием создания (инициализации) данного объекта. В противном случае этот переход никак не помечается. Если этот переход не помечен, то он является первым переходом на диаграмме состояний в следующее за ним состояние. Каждая диаграмма или под-диаграмма состояний должна иметь единственное начальное состояние.

Конечное состояние (final state) – разновидность псевдосостояния, обозначающее прекращение процесса изменения состояний конечного автомата или нахождения моделируемого объекта в составном состоянии.

В этом состоянии должен находиться моделируемый объект или система по умолчанию после завершения работы конечного автомата. Оно служит для указания на диаграмме состояний графической области, в которой завершается процесс изменения состояний или жизненный цикл данного объекта.

Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 3, б), в которую может только входить стрелка-переход. Каждая диаграмма состояний или подсостояний может иметь несколько конечных состояний, при этом все они считаются эквивалентными на одном уровне вложенности состояний.

Переход (transition) – отношение между двумя состояниями, которое указывает на то, что объект в первом состоянии должен выполнить определенные действия и перейти во второе состояние.

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события, а также действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое.

Событие (event) – спецификация существенных явлений в поведении системы, которые имеют местоположение во времени и пространстве.

Формально, событие представляет собой спецификацию факта, имеющего место в пространстве и во времени. Про события говорят, что они «происходят», при этом отдельные события должны быть упорядочены во времени. После наступления события нельзя уже вернуться к предыдущим, если такая возможность явно не предусмотрена в модели.

Семантика понятия события фиксирует внимание на внешних проявлениях качественных изменений, происходящих при переходе моделируемого объекта из состояния в состояние.

Переход называется **триггерным**, если его специфицирует событие-триггер, связанное с внешними условиями по отношению к рассматриваемому состоянию.

В этом случае рядом со стрелкой триггерного перехода обязательно указывается имя события в форме строки текста, начинающейся со строчной буквы. Наиболее часто в качестве имен триггерных переходов задают имена операций, вызываемых у тех или иных объектов системы. После имени такого события следуют круглые скобки для явного задания параметров соответствующей операции. Если таких параметров нет, то список параметров со скобками может отсутствовать. Например, переход на рис. 4, а, является триггерным, поскольку с ним связано конкретное событие-триггер, происходящее асинхронно при срабатывании некоторого датчика.

Переход называется **нетриггерным**, если он происходит по завершении выполнения ду-деятельности в данном состоянии.

Нетриггерные переходы часто называют переходами по завершении ду-деятельности. Для них рядом со стрелкой перехода не указывается никакого имени события, а в исходном состоянии должна быть описана внутренняя ду-деятельность, по окончании которой произойдет тот или иной нетриггерный переход.

Рисунок 219 – Графическое изображение триггерного (а) и нетриггерного (б) переходов на диаграмме состояний

Сторожевое условие (guard condition) – логическое условие, записанное в прямых скобках и представляющее собой булевское выражение. При этом булевское выражение должно принимать одно из двух взаимно

исключающих значений: «истина» или «ложь». Из контекста диаграммы состояний должна явно следовать семантика этого выражения, а для записи выражения может использоваться обычный язык, псевдокод или язык программирования.

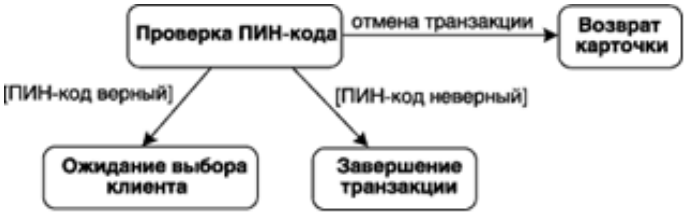


Рисунок 220 – Триггерные и нетриггерные переходы на диаграмме состояний

Изображенный фрагмент диаграммы состояний (рис. 5) моделирует изменение состояний банкомата при проверке ПИН-кода. Нетриггерные переходы на данной диаграмме помечены сторожевыми условиями, которые исключают конфликт между ними. Что касается триггерного перехода, помеченного событием отмена транзакции, то он происходит независимо от проверки ПИН-кода в том случае, когда клиент решил отказаться от ввода ПИН-кода.

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

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

В качестве примера выражения действия перехода (рис. 6) может служить отображение сообщения на экране банкомата в том случае, когда запрашиваемая клиентом сумма превосходит остаток на его счету. В случае если кредит не превышен, то происходит переход в состояние получения наличных.

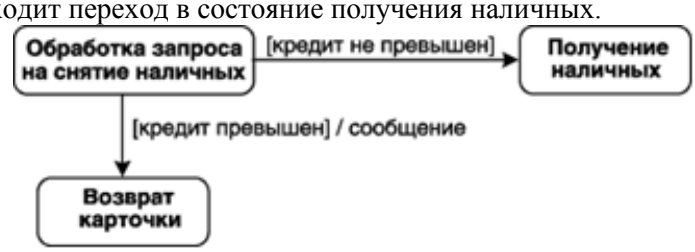


Рисунок 221 – Выражение действия перехода на диаграмме состояний

Составное состояние (composite state) – сложное состояние, которое состоит из других вложенных в него состояний (состояние-композит). Вложенные состояния выступают по отношению к составному состоянию как **подсостояния** (substate). И хотя между ними имеет место отношение композиции, графически все вершины диаграммы, которые соответствуют вложенным состояниям, изображаются внутри символа составного состояния (рис. 7). В этом случае размеры графического символа составного состояния увеличиваются, так чтобы вместить в себя все подсостояния.

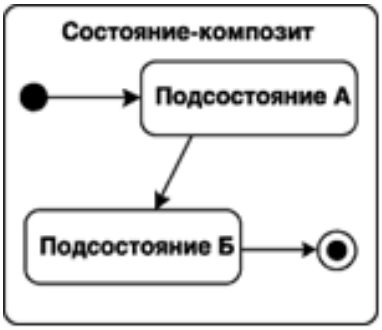


Рисунок 222 – Графическое представление составного состояния с двумя вложенными в него последовательными подсостояниями

Составное состояние может содержать или несколько последовательных подсостояний, или несколько параллельных конечных подавтоматов. Каждое состояние-композит может уточняться только одним из указанных способов. При этом любое из подсостояний, в свою очередь, может быть состоянием-композитом и содержать внутри себя другие вложенные подсостояния. Количество уровней вложенности составных состояний в языке UML не фиксировано.

Последовательные подсостояния (sequential substates) – вложенные состояния состояния-композита, в рамках которого в каждый момент времени объект может находиться в одном и только одном подсостоянии.

Поведение объекта в этом случае представляет собой последовательную смену подсостояний, от начального до конечного. Моделируемый объект или система продолжает находиться в составном состоянии, тем не менее, введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты его внутреннего поведения.

В качестве примера моделируемой системы стоит рассмотреть обычный телефонный аппарат. Он может находиться в различных состояниях, в частности в состоянии дозвона до абонента. Очевидно, для того чтобы позвонить, необходимо снять телефонную трубку, услышать тоновый сигнал, после чего набрать нужный телефонный номер. Таким образом, состояние дозвона до абонента является составным и состоит из двух последовательных подсостояний: Телефонная трубка поднята и Набор телефонного номера. Фрагмент диаграммы состояний для этого примера содержит одно состояние-композит, которое состоит из двух последовательных подсостояний (рис. 8).

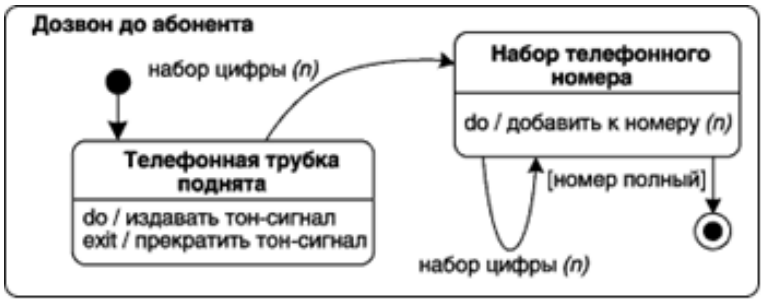


Рисунок 223 – Пример составного состояния с двумя вложенными последовательными подсостояниями

Некоторых пояснений могут потребовать переходы. Два из них специфицируют событие-триггер, которое имеет имя: набор цифры(n) с параметром n. В качестве параметра, как нетрудно предположить, выступает отдельная цифра на диске телефонного аппарата. Переход из начального подсостояния не содержит никакой строки текста. Последний переход в конечное подсостояние также не имеет события-триггера, но имеет сторожевое условие, проверяющее полноту набранного номера абонента. Только в случае истинности этого условия телефонный аппарат может перейти в конечное состояние для состояния-композита Дозвон до абонента.

Каждое составное состояние должно содержать в качестве вложенных состояний начальное и конечное состояния. При этом начальное подсостояние является исходным, когда происходит переход объекта в данное составное состояние. Если составное состояние содержит внутри себя конечное состояние, то переход в это вложенное конечное состояние означает завершение нахождения объекта в данном составном состоянии. Важно помнить, что для последовательных подсостояний начальное и конечное состояния должны быть единственными в каждом составном состоянии.

Это можно объяснить следующим образом. Каждая совокупность вложенных последовательных подсостояний представляет собой конечный подавтомат того конечного автомата, которому принадлежит рассматриваемое составное состояние. Поскольку каждый конечный автомат может иметь по определению единственное начальное и единственное конечное состояния, то для любого его конечного подавтомата это условие также должно выполняться.

Параллельные подсостояния (concurrent substates) – вложенные состояния, используемые для спецификации двух и более конечных подавтоматов, которые могут выполняться параллельно внутри составного состояния.

Каждый из конечных подавтоматов занимает некоторую графическую область внутри составного состояния, которая отделяется от остальных горизонтальной пунктирной линией. Если на диаграмме состояний имеется составное состояние с вложенными параллельными подсостояниями, то объект может одновременно находиться в каждом из этих подсостояний.

Отдельные параллельные подсостояния могут, в свою очередь, состоять из нескольких последовательных подсостояний (рис. 9). В этом случае по определению моделируемый объект может находиться только в одном из последовательных подсостояний каждого подавтомата. Таким образом, для фрагмента диаграммы состояний (рис. 9) допустимо одновременное нахождение объекта только в следующих подсостояниях: (А, В, Г), (Б, В, Г), (А, В, Д), (Б, В, Д).

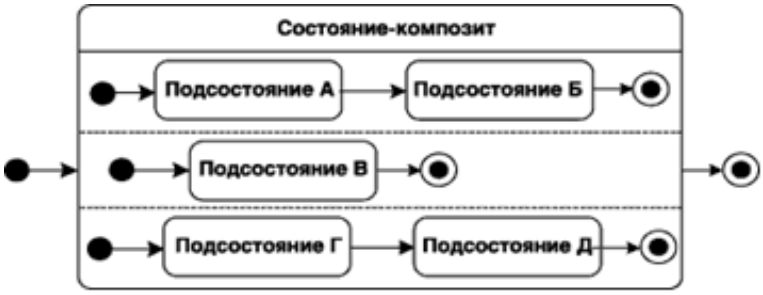


Рисунок 224 – Графическое изображение состояния-композита с вложенными параллельными подсостояниями

Несовместимое подсостояние (disjoint substate) – подсостояние, в котором подсистема не может находиться одновременно с другими подсостояниями одного и того же составного состояния.

В этом контексте недопустимо нахождение объекта одновременно в несовместимых подсостояниях (А, Б, В) или (В, Г, Д).

Может оказаться необходимым учесть ту часть деятельности, которая была выполнена на момент выхода из этого состояния-композита, чтобы не начинать ее выполнение сначала. Для этой цели в языке UML существует историческое состояние.

Историческое состояние (history state) – псевдосостояние, используемое для запоминания того из последовательных подсостояний, которое было текущим в момент выхода из составного состояния.

Историческое состояние применяется только в контексте составного состояния. При этом существует две разновидности исторического состояния: неглубокое или недавнее и глубокое или давнее (рис. 10).

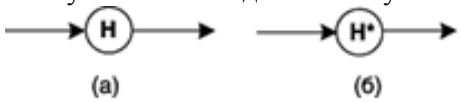


Рисунок 225 – Графическое изображение недавнего (а) и давнего (б) исторического состояния

Неглубокое историческое состояние (shallow history state) обозначается в форме небольшой окружности, в которую помещена латинская буква «H» (рис. 10, а). Это состояние обладает следующей семантикой. Во-первых, оно является первым подсостоянием в составном состоянии, и переход извне в рассматриваемое составное состояние должен вести непосредственно в данное историческое состояние. Во-вторых, при первом попадании в неглубокое историческое состояние оно не хранит никакой истории. Другими словами, при первом переходе в недавнее историческое состояние оно заменяет собой начальное состояние соответствующего конечного подавтомата.

Далее могут последовательно изменяться вложенные подсостояния. Если в некоторый момент происходит выход из составного состояния (например, в случае наступления некоторого события), то рассматриваемое историческое состояние запоминает то из подсостояний, которое было текущим на момент выхода из данного составного состояния. При последующем входе в это составное состояние неглубокое историческое подсостояние имеет непустую историю и сразу отправляет конечный подавтомат в запомненное подсостояние, минуя все предшествующие ему подсостояния.

Глубокое историческое состояние (deep history state или состояние глубокой истории) также обозначается в форме небольшой окружности, в которую помещена латинская буква «H» с дополнительным символом "*" (рис. 10, б), и служит для запоминания всех подсостояний любого уровня вложенности для исходного составного состояния.

В отдельных случаях возникает необходимость явно показать ситуацию, когда переход может иметь несколько исходных состояний или целевых состояний. Такой переход получил название – **параллельный переход.** Введение в рассмотрение параллельных переходов может быть обусловлено необходимостью синхронизировать и/или разделить отдельные процессы управления на параллельные нити без спецификации дополнительной синхронизации в параллельных конечных подавтоматах.

Графически такой переход изображается вертикальной черточкой, аналогично обозначению перехода в известном формализме сетей Петри. Если параллельный переход имеет две или более исходящих из него дуг (рис. 11, а), то его называют разделением (fork). Если же он имеет две или более входящие дуги (рис. 11, б), то его называют слиянием (join). Текстовая строка спецификации параллельного перехода записывается рядом с черточкой и относится ко всем входящим или исходящим дугам.

Рисунок 226 – Графическое изображение перехода-разделения в параллельные подсостояния (а) и перехода-слияния из параллельных подсостояний (б)

Состояние синхронизации (synch state) – псевдосостояние в конечном автомате, которое используется для синхронизации параллельных областей конечного автомата.

Синхронизирующее состояние обозначается небольшой окружностью, внутри которой помещен символ звездочки "*". Оно используется совместно с переходом-слиянием или переходом-разделением для того, чтобы явно указать события в других конечных подавтоматах, оказывающие непосредственное влияние на поведение данного подавтомата.

Так, например, при включении компьютера с некоторой сетевой операционной системой параллельно начинается выполнение нескольких процессов. В частности, происходит проверка пароля пользователя и запуск различных служб. При этом работа пользователя на компьютере станет возможной только в случае успешной его аутентификации, в противном случае компьютер может быть выключен.

Рассмотренные особенности синхронизации этих параллельных процессов учтены на соответствующей диаграмме состояний с помощью синхронизирующего состояния (рис. 12).

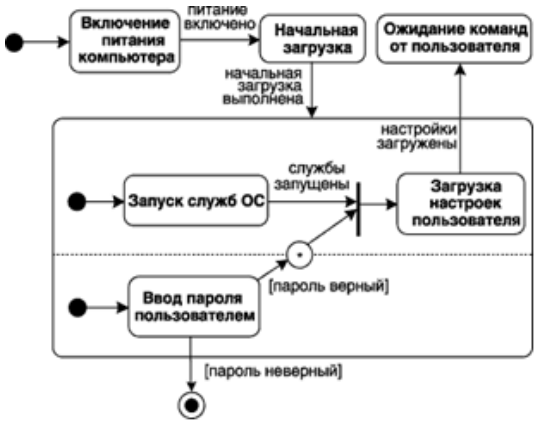


Рисунок 227 – Диаграмма состояний для примера включения компьютера

Пример: Программное средство представляет собой базу данных и обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствие с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

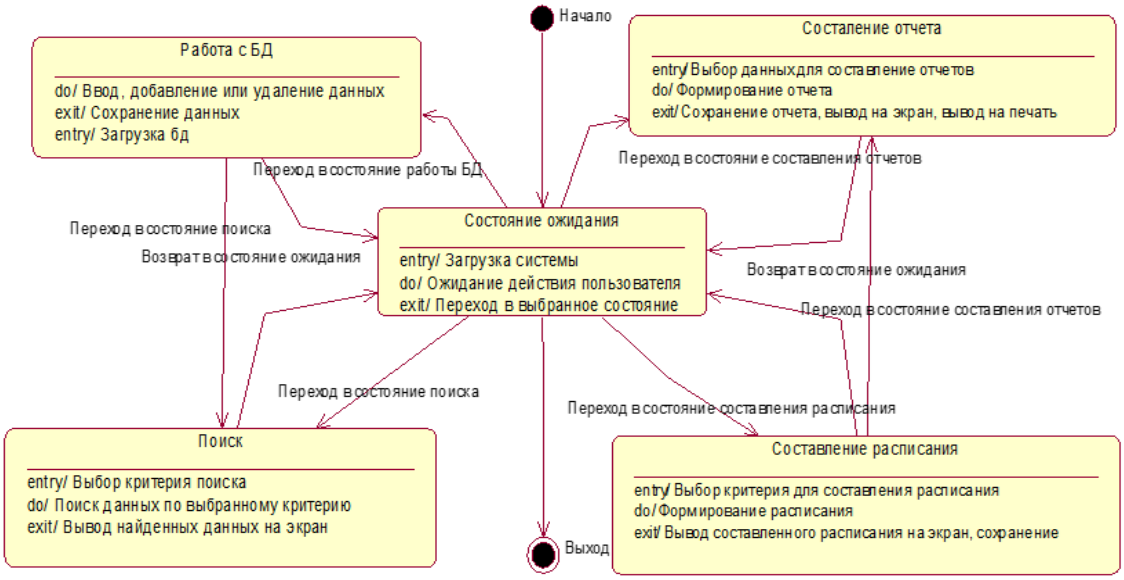


Рисунок 228 – Пример диаграммы состояний

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите Visual Paradigm for UML CE.
2. Откройте проект, содержащий диаграмму вариантов использования, построенную в практическом занятии №13.
3. Для добавления новой диаграммы достаточно в меню Diagram > New Diagram выбрать тип диаграммы State Machine Diagram и нажмите кнопку Next. Задаnm имя диаграммы и нажать OK.
4. Либо, в ранее созданной диаграмме классов, щелкнуть правой кнопкой мыши по классу, для которого необходимо создать диаграмму состояний и в контекстном меню выбрать **Sub Diagrams > 1) New Diagram > State Machine Diagram >** при необходимости изменить имя диаграммы и нажать OK.

2) Existing Diagram > выбрать в списке ранее созданную диаграмму и добавить ее в качестве поддиаграммы.

5. В результате отобразится рабочая область с элементами для построения диаграммы состояний (рис. 14).

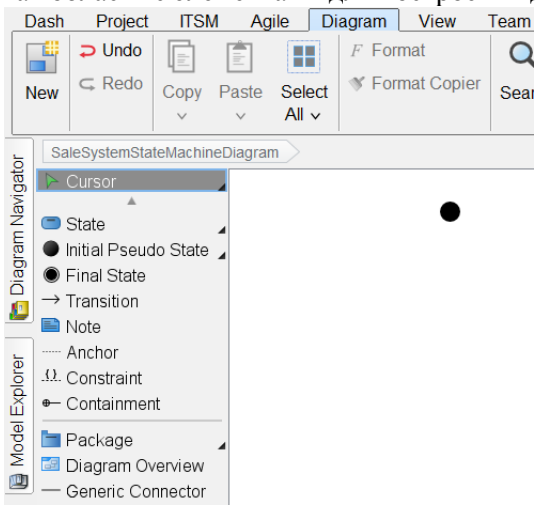


Рисунок 229 – Рабочая область для построения диаграммы состояний

Далее будут описаны принципы построения диаграммы состояний.

Для создания состояния необходимо на панели инструментов выбрать State и затем курсором мыши выбрать место расположения на диаграмме (Рисунок 15). После создания класса необходимо задать его имя (Рисунок 16).

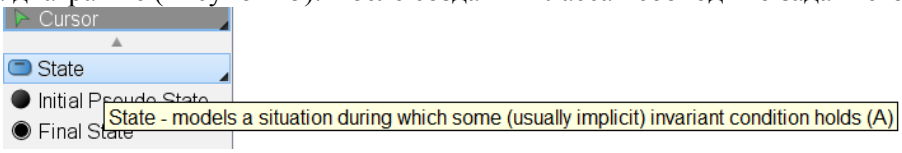


Рисунок 230 – Создание состояния

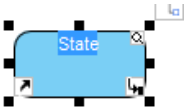


Рисунок 231 – Задание имени в созданном состоянии

Для создания перехода между на начальным состоянием и последующим состоянием или между состояниями выберите **Transition** > **State (**Рисунок 17**).**

Перетащите курсор мыши на пустое место на диаграмме для создания нового или на уже существующее состояние для соединения. Отпустите кнопку мыши (Рисунок 17)**.**

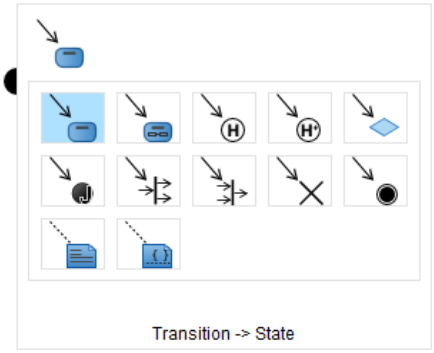


Рисунок 232 – Выбор типа перехода

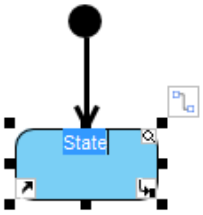


Рисунок 233 – Создание перехода

Для добавления внутренних действий, нажмите правой кнопкой мыши по состоянию и выберите пункт **Open Specification…** добавить действия можно посредством определения значений полей **Entry, Do, Exit Activity** (Рисунок 19). Либо выделите состояние и нажмите **Enter** (рисунок 19).

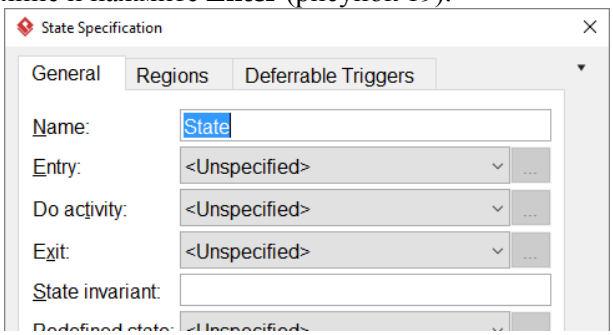


Рисунок 234 – Изменение внутренних действий

Например, для добавления внутреннего действия **Entry** щелкните раскрывающийся список и выберите **Create Activity…** (рисунок 20).

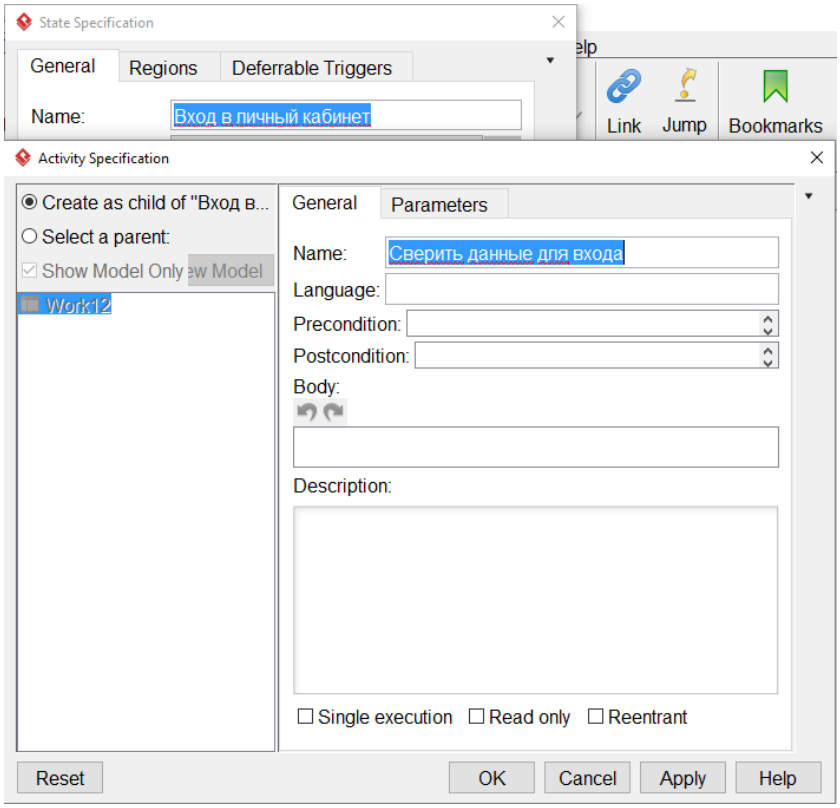


Рисунок 235 – Создание внутреннего действия

Для добавления описания перехода дважды щелкните по нему и введите описание (рисунок 21).

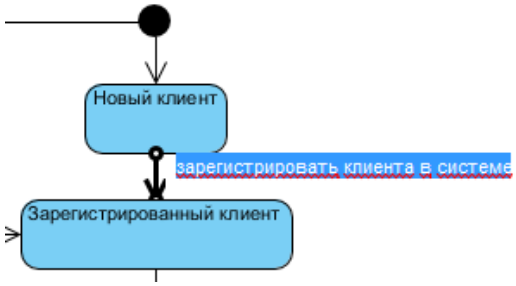


Рисунок 236 – Добавление описания перехода

Чтобы добавить переход разветвления или объединения наведите курсор в правый нижний угол элемента **Initial Pseudo State** и выберите **Fork** или **Join** (рисунок 22). После добавления элемента, для того чтобы изменить ориентацию на горизонтальную, щелкните правой кнопкой мыши по элементу и в контекстном меню выберите **Orientation > Horizontal.**

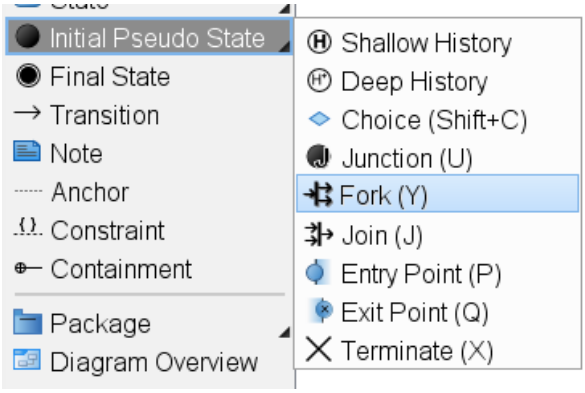


Рисунок 237 – Добавление перехода разветвления

Для того чтобы вернуться к диаграмме классов, нажмите на имя класса в иерархии диаграмм (рисунок 22).

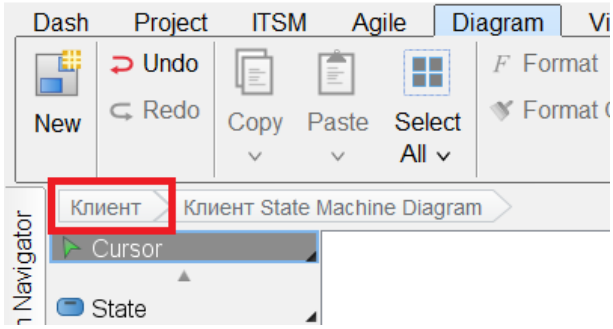


Рисунок 238 – Открытие диаграммы классов

Основываясь, на описанных выше принципах построения, постройте диаграммы состояний для системы продажи товаров по каталогу в соответствии с рисунками 24-27.

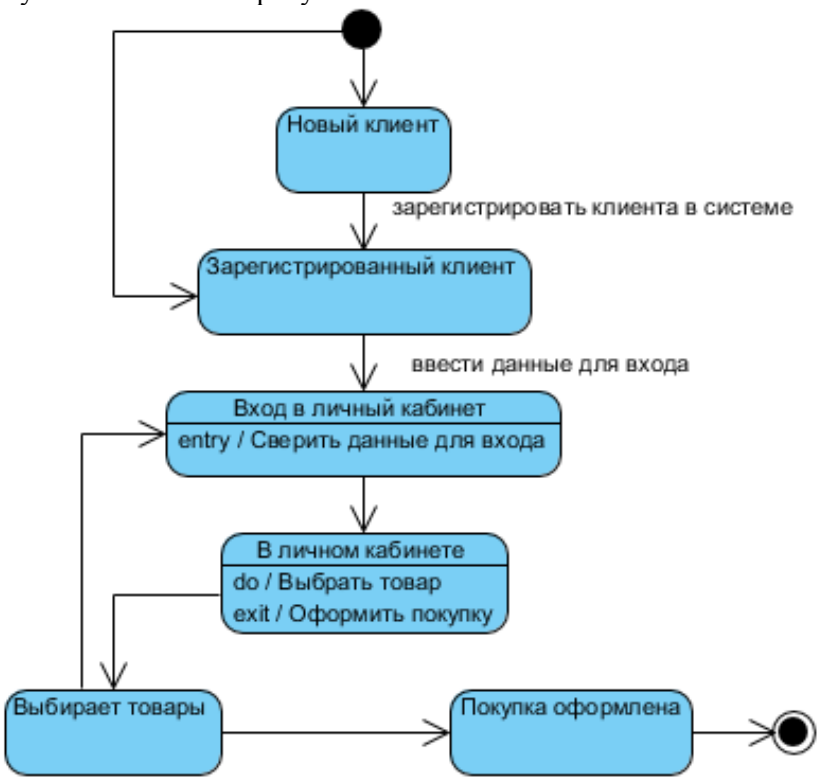


Рисунок 239 – Диаграмма состояний для класса «Клиент»

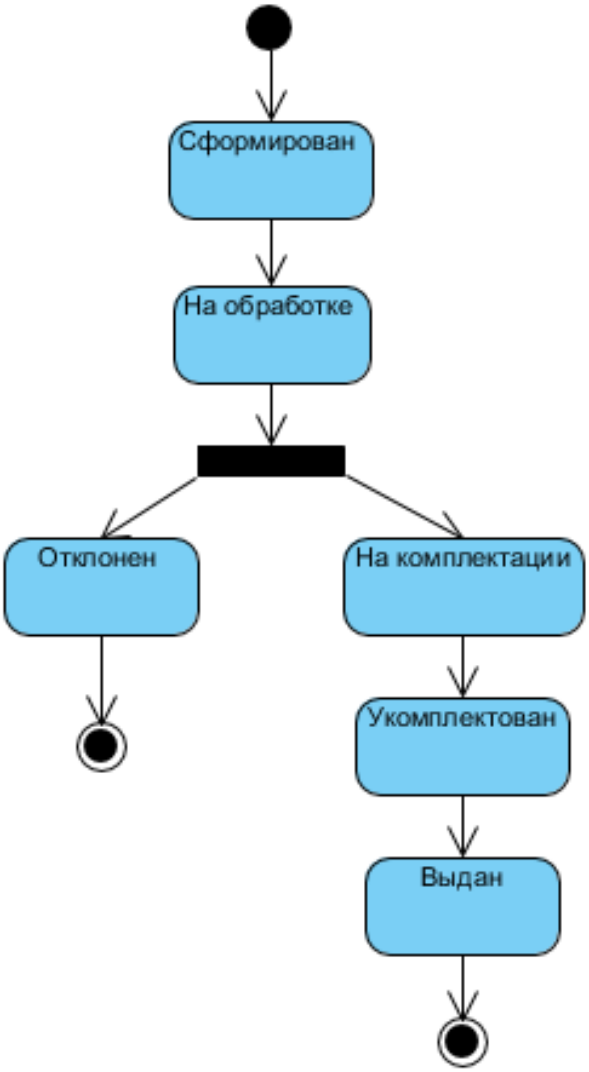


Рисунок 240 – Диаграмма состояний для класса «Заказ»

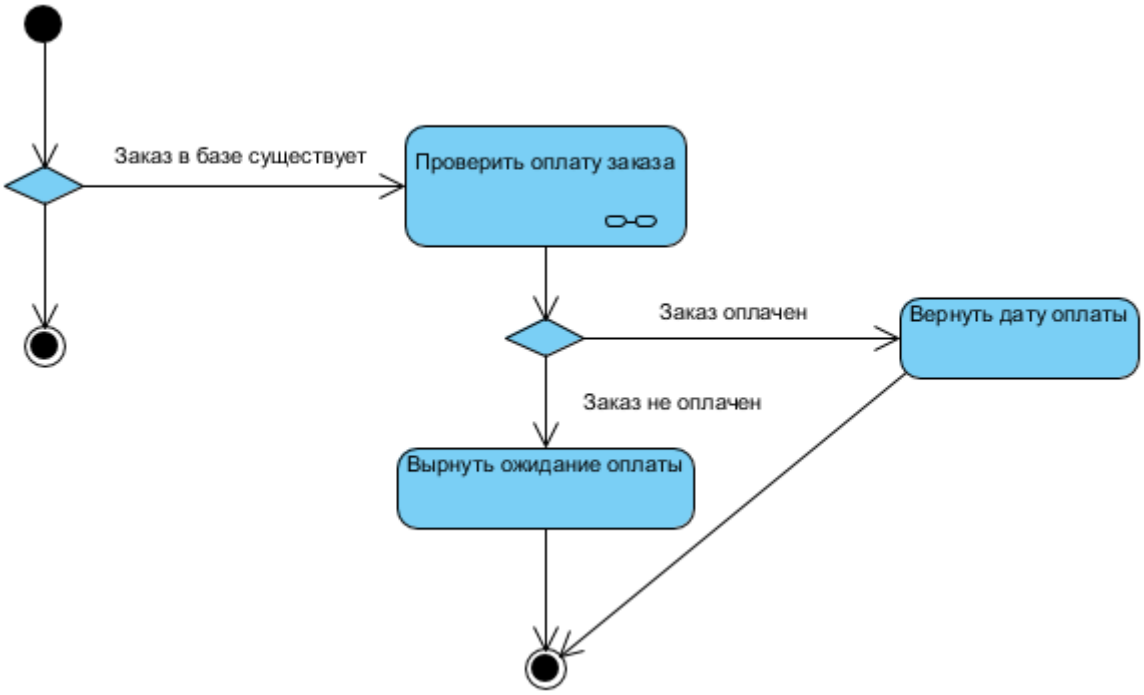


Рисунок 241 – Диаграмма состояний для класса «Заказ_Оплата»

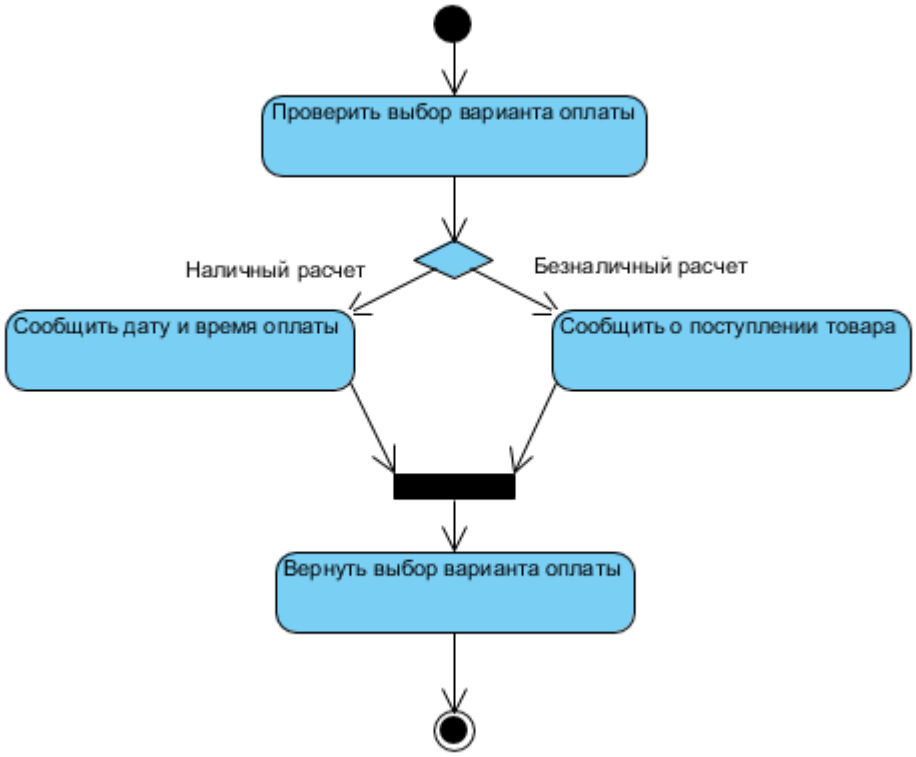


Рисунок 242 – Диаграмма состояний для класса «Вариант_Оплаты»

5. Задание

Построить диаграммы состояний для всех классов диаграммы классов, построенной в практическом занятии №13 в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Дайте определение понятию «диаграмма состояний».
2. Опишите назначение диаграммы состояний.
3. Дайте определение понятию «конечный автомат».
4. Дайте определение понятиям «состояния», «действие», «псевдосостояние». Графическое изображение состояния.
5. Что представляет собой переход? Какие бывают переходы, в чем их различие?
6. Дайте определение понятию «событие».
7. Дайте определения следующим понятиям: «составное состояние», «последовательные подсостояния», «параллельные подсостояния», «несовместимое подсостояние», «историческое состояние», «параллельный переход», «состояние синхронизации».

Практическая работа №15

Разработка диаграмм деятельности с помощью Visual Paradigm for UML

1. Цель работы

Целью работы является изучение основ создания диаграмм деятельности на языке UML с помощью case-средства Visual Paradigm for UML CE, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

2. Задачи

Основными задачами практической работы являются:

– получить навыки создания диаграмм деятельности с помощью case-средства Visual Paradigm for UML CE.

3. Краткие теоретические сведения

При моделировании поведения проектируемой или анализируемой программной системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и процедурной реализации выполняемых системой операций. Для этой цели, как правило, используются блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных процедур или элементарных операций, которые в совокупности приводят к получению желаемого результата. С увеличением сложности системы строгое соблюдение определенной последовательности выполняемых действий приобретает большое значение.

Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления деятельности и действий, а также в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некой операции, а переход в следующее состояние происходит только после завершения выполнения этой операции. Диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия или деятельности, а дугами – переходы от одного состояния действия к другому.

Диаграммы деятельности (activity diagram) – частный случай диаграмм состояний. Они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних действий и деятельности. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции определенного класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML **деятельность** представляет собой совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к результату или действию. На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения. Диаграмма деятельности предназначена для моделирования поведения систем, хотя время в явном виде отсутствует на этой диаграмме. Ситуация здесь во многом аналогична диаграмме состояний, однако имеет ряд особенностей.

Состояние деятельности (activity state) – состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определенного времени. Переход из состояния деятельности происходит после выполнения специфицированной в нем ду-деятельности, при этом ключевое слово do в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным.

Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

Состояние деятельности можно представлять себе как составное состояние, поток управления которого включает только другие состояния деятельности и действий.

Рисунок 243 – Графическое изображение состояний деятельности действия

Состояние действия (action state) – специальный случай состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Переход из состояния действия происходит после завершения входного действия. Состояние действия не может иметь внутренних переходов, поскольку оно

является элементарным. Использование состояния действия заключается в моделировании шага выполнения алгоритма. Графически состояния деятельности и действия изображаются одинаковой фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами. Внутри этой фигуры записывается имя состояния деятельности или действия в форме выражения, которое должно быть уникальным в пределах одной диаграммы деятельности.

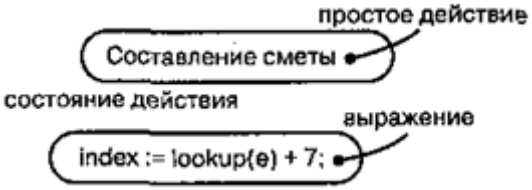


Рисунок 244 – Графическое изображение состояний действия на диаграмме деятельности

Действие может быть записано на естественном языке, псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами (рис. 2). Если же действие может быть представлено в формальном виде, то оно записывается на том языке программирования, на котором предполагается реализовывать разрабатываемый проект.

Когда возникает необходимость представить на диаграмме деятельности сложное действие, состоящее из нескольких более простых, то используют специальное обозначение так называемого состояния под-деятельности. **Состояние под-деятельности** (subactivity state) – состояние в графе деятельности, которое служит для представления неатомарной последовательности шагов процесса. Это состояние является графом деятельности и обозначается специальной пиктограммой в правом нижнем углу символа состояния действия (рис. 3). Данная конструкция может применяться к любому элементу языка UML, который поддерживает «вложенность» своей структуры. При этом пиктограмма может быть дополнительно помечена типом вложенной структуры.

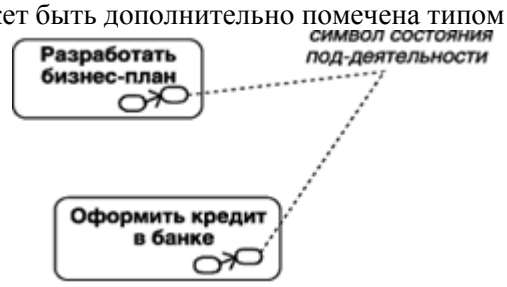


Рисунок 245 – Графическое изображение состояния под-деятельности

Каждая диаграмма деятельности должна иметь единственное начальное и конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний. При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии.

Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз или слева направо. В этом случае начальное состояние будет изображаться в верхней или левой части диаграммы, а конечное – в ее нижней или правой части. В интересах удобства визуального представления на диаграмме деятельности допускается изображать несколько конечных состояний. В этом случае все их принято считать эквивалентными друг другу.

Переход на диаграмме деятельности аналогичен переходу на диаграмме состояний. При построении диаграммы деятельности используются только нетриггерные переходы, т.е. такие, которые происходят сразу после завершения деятельности или выполнения соответствующего действия. Такой переход передает управление в последующее состояние сразу, как только закончится действие или деятельность в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то его можно никак не помечать. Если же таких переходов несколько, то при моделировании последовательной деятельности запускается только один из них. В этом случае для каждого из таких переходов должно быть явно записано собственное сторожевое условие в прямых скобках.

При этом для всех выходящих из некоторого состояния деятельности переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения промежуточного результата. Такая ситуация получила название **ветвления**, а для ее обозначения применяется специальный символ решения.

Графически ветвление на диаграмме деятельности обозначается символом решения (decision), изображаемого в форме небольшого ромба, внутри которого нет никакого текста (рис. 4). В ромб может входить только одна стрелка от того состояния действия, после выполнения, которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине

символа решения. Выходящих стрелок может быть две или более. Для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

Для графического объединения альтернативных ветвей на диаграмме деятельности рекомендуется использовать аналогичный символ в форме ромба, который в этом случае называют **соединением** (merge). Наличие этого символа, внутри которого также не записывается никакого текста, упрощает визуальный контроль логики выполнения процедурных действий на диаграмме деятельности (рис. 4 внизу). Входящих стрелок у символа соединения может быть несколько, они исходят от состояний действия, принадлежащих к одной из взаимно исключающих ветвей. Выходить из ромба соединения может только одна стрелка, при этом ни входящие, ни выходящая стрелки не должны содержать сторожевых условий. Исключением является ситуация, когда с целью сокращения диаграммы объединяют символ решения с символом соединения.

Пример: моделируется ситуация, возникающая в супермаркетах при оплате товаров. Как правило, заплатить за покупки можно либо наличными, либо по кредитной карточке. Если покупателем выбран вариант оплаты по кредитной карточке, то проверяется сумма баланса предъявленной к оплате кредитной карточки. При этом оплата происходит только в том случае, если общая стоимость приобретаемых товаров не превышает суммы баланса этой карточки. В противном случае оплаты не происходит, и товар остается у продавца.

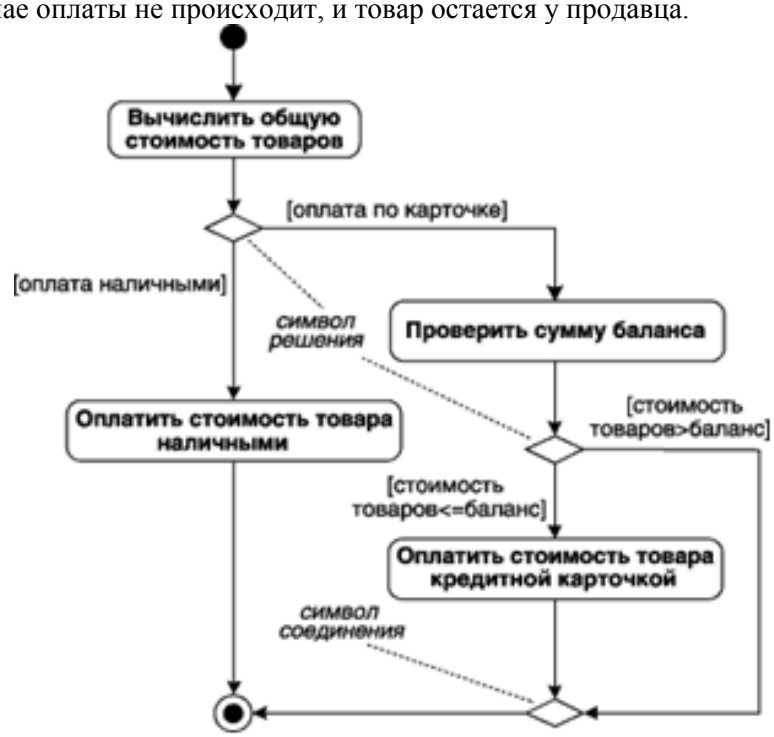


Рисунок 246 – Различные варианты ветвлений на диаграмме деятельности

На диаграмме деятельности изображаются параллельные процессы, поскольку распараллеливание вычислений существенно повышает быстродействие программных систем. В диаграммах деятельности для изображения параллельных процессов используется специальный символ.

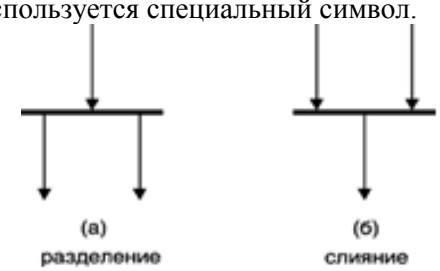


Рисунок 247 – Графическое изображение разделения и слияния параллельных потоков управления на диаграмме деятельности

На диаграммах деятельности такая черточка изображается отрезком горизонтальной, реже – вертикальной, линии, толщина которой несколько шире линий простых переходов диаграммы деятельности. При этом **разделение** (fork) имеет один входящий переход и несколько выходящих (рис. 6, а), которые изображаются отрезками вертикальных, реже – горизонтальных, линий. **Слияние** (join), наоборот, имеет несколько входящих переходов и один выходящий (рис. 6, б). Параллельные переходы на диаграмме деятельности можно изображать в удлиненной форме, а входящие и выходящие переходы вертикальными стрелками.

Пример: Регистрация пассажиров в аэропорту. Первоначально выполняется деятельность по проверке билета: если билет не действителен, он возвращается пассажиру; если билет действителен, то пассажиру выдается посадочный талон, в дополнение проверяется гражданство и наличие багажа у пассажира. Если есть багаж, то его проверка может быть выполнена параллельно, после чего пассажиру выдается талон на багаж. Если пассажир является иностранным гражданином, то дополнительно проверяется наличие у него визы. Если виза действительна,

то проверка завершается успешно, и пассажир может проследовать на посадку. Если же виза не действительна, то для этого пассажира посадка оказывается невозможной, и ему не выдается посадочный талон и талон на багаж. Происходит прекращение всех выполняемых сотрудниками аэропорта действий.

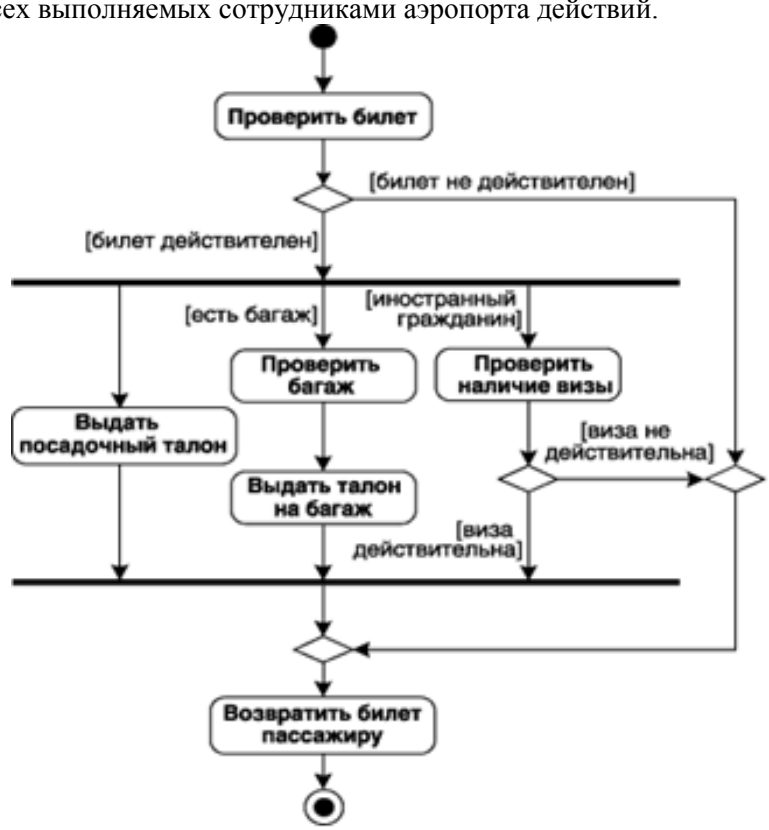


Рисунок 248 – Диаграмма деятельности для примера регистрации пассажиров в аэропорту

Дорожка (swimlane) – графическая область диаграммы деятельности, содержащая элементы модели, ответственность за выполнение которых принадлежит отдельным подсистемам.

В данном случае имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму деятельности сверху. При этом все состояния на диаграмме деятельности делятся на группы, разграниченные вертикальными линиями.

Две соседних линии и образуют дорожку, а группа состояний между этими линиями выполняется организационным подразделением (отделом, группой, отделением, филиалом) или сотрудником компании (рис. 7). В последнем случае принято указывать должность сотрудника, ответственного за выполнение определенных действий.

Названия подразделений или должностей явно указываются в верхней части дорожки. Пересекать линию дорожки могут только переходы, которые в этом случае обозначают выход или вход потока управления в соответствующее подразделение компании. Порядок следования дорожек не несет какой-либо семантической информации и определяется соображениями удобства.

Рисунок 249 – Вариант диаграммы деятельности с дорожками

Пример: фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов в форме заказов. Подразделениями компании обычно являются отдел приема и оформления заказов, отдел продаж и склад. Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения. В этом случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое подразделение торговой компании должно выполнять то или иное действие (рис. 8).

После принятия заказа от клиента отделом приема и оформления заказов осуществляется распараллеливание деятельности на два потока (переход-разделение). Первый из них остается в этом же отделе и связан с получением оплаты от клиента за заказанный товар. Второй инициирует выполнение действия по регистрации заказа в отделе продаж (модель товара, размеры, цвет, год выпуска и пр.). Однако выдача товара со склада начинается только после того, как будет получена от клиента оплата за товар (переход-слияние). Затем выполняется подготовка товара к отправке и его отправка клиенту в отделе продаж. После завершения этих деятельностей заказ закрывается в отделе приема и оформления заказов.

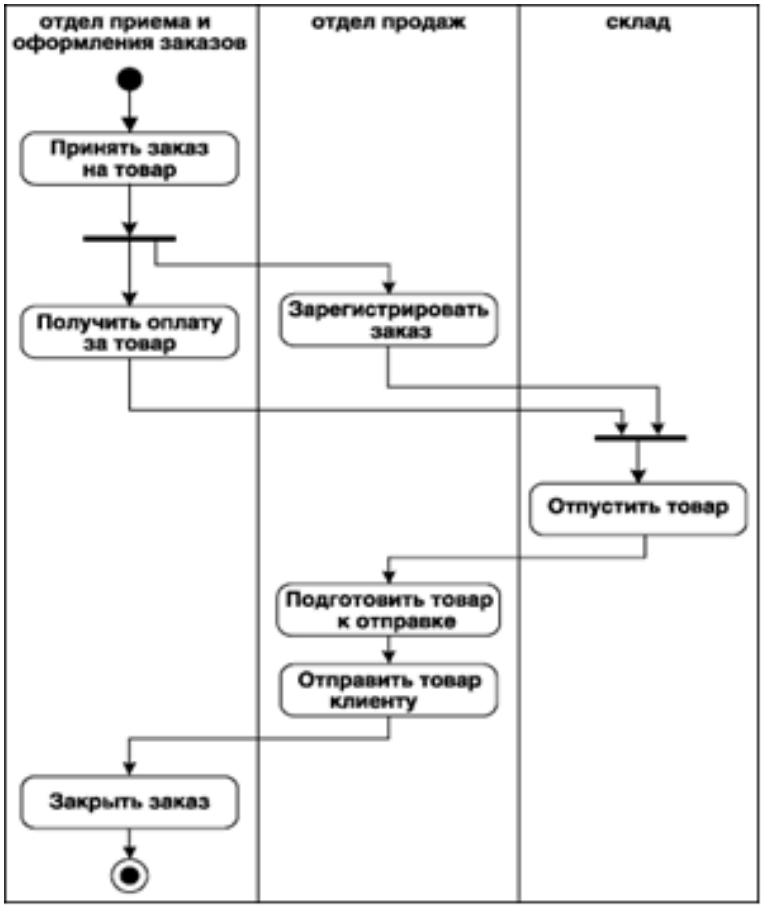


Рисунок 250 – Фрагмент диаграммы деятельности для торговой компании

Действия на диаграмме деятельности могут производиться над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

На диаграмме деятельности с дорожками расположение объекта может иметь дополнительный смысл. А именно, если объект расположен на границе двух дорожек, то это может означать, что переход к следующему состоянию действия в соседней дорожке ассоциирован с нахождением документа в некотором состоянии. Если же объект расположен внутри дорожки, то и состояние этого объекта целиком определяется действиями данной дорожки.

В примере с торговой компанией центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до обращения клиента заказ как объект отсутствует и возникает после контакта с клиентом. В результате фиксируется полученный заказ, потом он регистрируется в отделе продаж, затем он передается на склад, где после получения оплаты за товар оформляется окончательно. После того, как товар отправлен клиенту, эта информация вносится в заказ, и он считается выполненным (рис. 9).

Достоинством диаграммы деятельности является возможность визуализировать отдельные аспекты поведения рассматриваемой системы или реализации отдельных операций классов в виде процедурной последовательности действий. Таким образом, полная модель системы может содержать одну или несколько диаграмм деятельности, каждая из которых описывает последовательность реализации либо наиболее важных вариантов использования (типичный ход событий и все исключения), либо нетривиальных операций классов.

Рисунок 251 – Фрагмент диаграммы деятельности торговой компании с объектом-заказом

Пример: Программное средство представляет собой базу данных «Автоматизация процесса составления расписания в учебном заведении». Программное средство обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствие с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

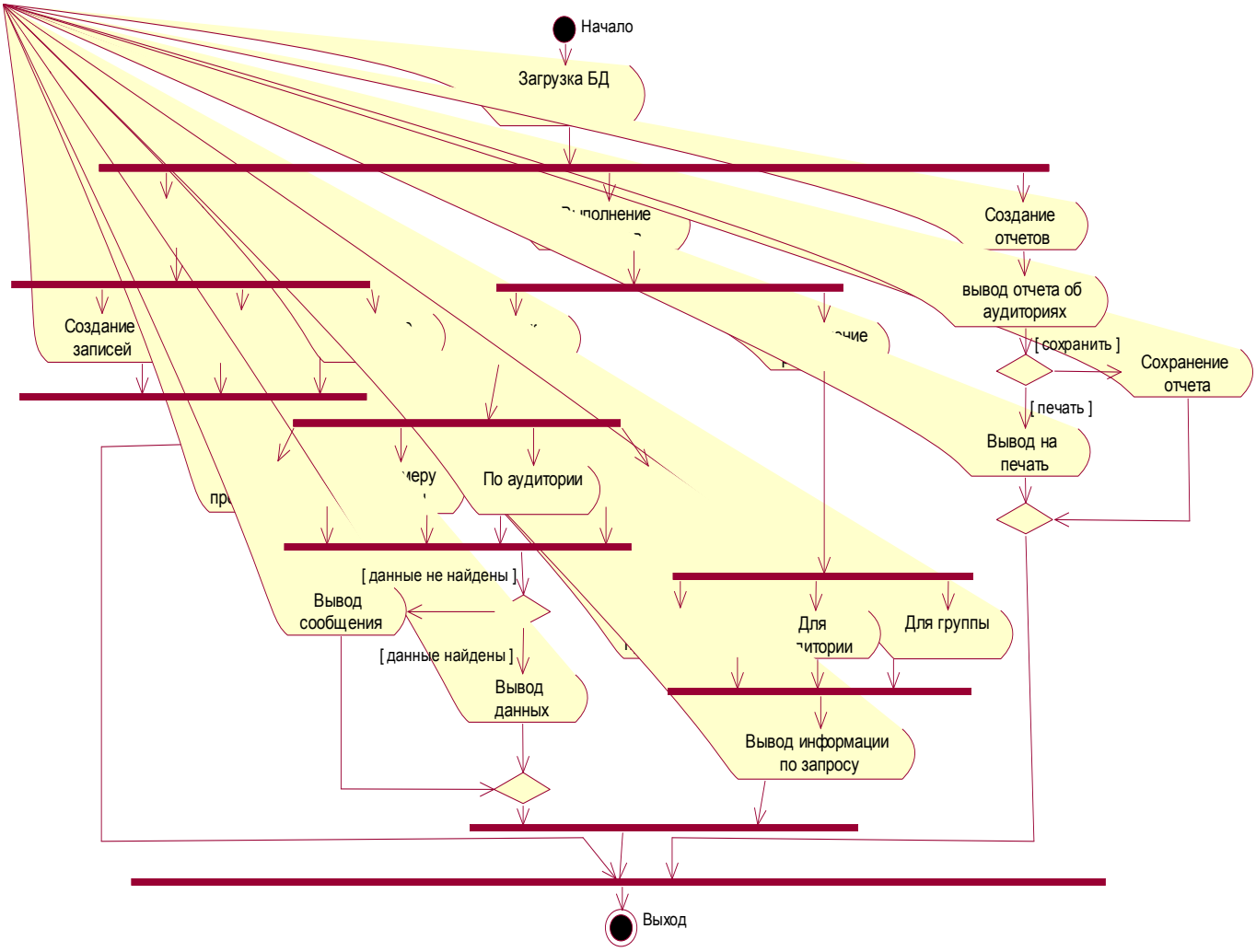


Рисунок 252 – Пример диаграммы деятельности

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите Visual Paradigm for UML CE.
2. Откройте проект, содержащий диаграмму вариантов использования, построенную в практическом занятии №14.
3. Для добавления новой диаграммы достаточно в меню Diagram > New Diagram выбрать тип диаграммы Activity Diagram и нажмите кнопку Next. Задать имя диаграммы SaleSystemActivityDiagram и нажать OK.
4. В результате отобразится рабочая область с элементами для построения диаграммы деятельности (рис. 11).

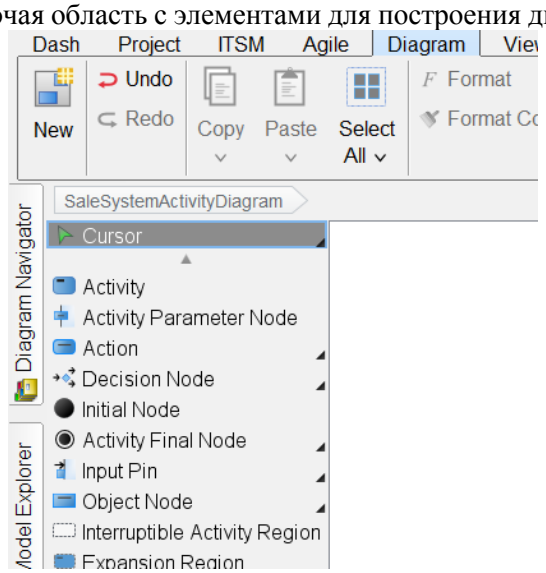


Рисунок 253 – Рабочая область для построения диаграммы деятельности

Далее будут описаны принципы построения диаграммы деятельности.

Для создания дорожек необходимо на панели инструментов выбрать Vertical SwimLine и затем курсором мыши выбрать место расположения на диаграмме (Рисунок 12). Чтобы добавить дорожку необходимо щелкнуть правой кнопкой мыши по элементу и в контекстном меню выбрать **Add Vertical Partition.** После создания дорожек необходимо задать их имена. Для это дважды щелкните по заголовку дорожки и введите имя (Рисунок 13).

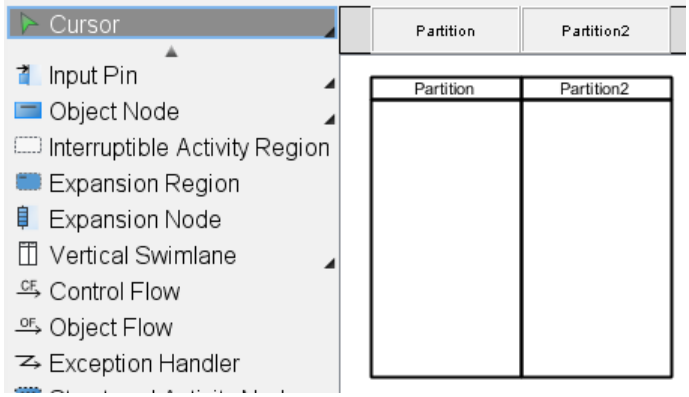


Рисунок 254 – Создание дорожек

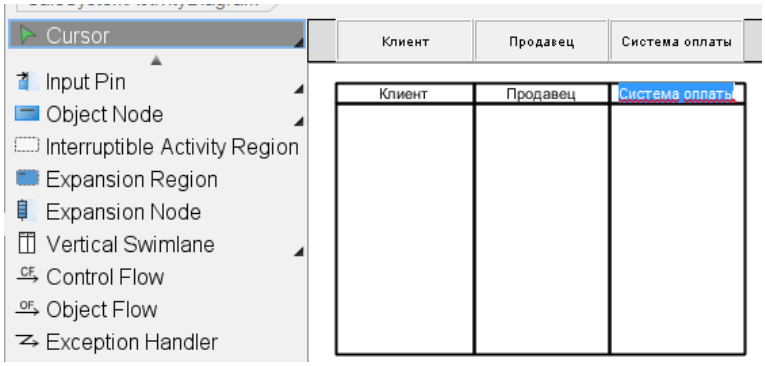


Рисунок 255 – Задание имени в созданных дорожках

Для добавления начального состояния на панели инструментов выберите **Initial Node** и поместите в нужное место рабочей области.

Для добавления нового состояния на панели инструментов выберите **Action** и поместите в нужное место рабочей области.

Для создания перехода между состояниями выберите **Control Flow > Action** (Рисунок 14).

Перетащите курсор мыши на пустое место на диаграмме для создания нового или на уже существующее состояние для соединения. Отпустите кнопку мыши (Рисунок 15)**.**

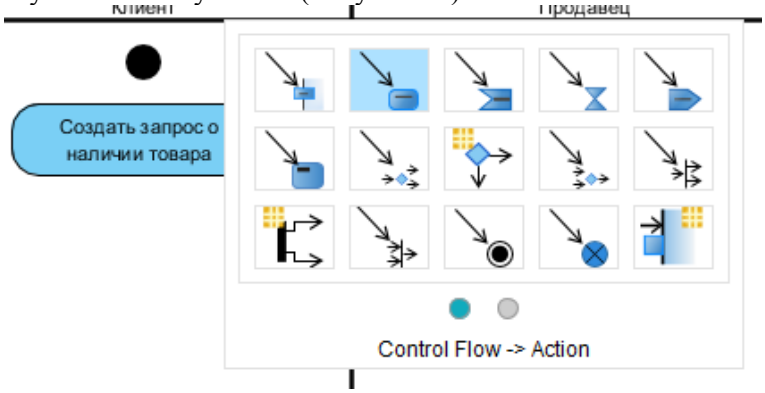


Рисунок 256 – Выбор типа перехода

Рисунок 257 – Создание перехода

Для добавления описания перехода дважды щелкните по нему и введите описание (рисунок 16).

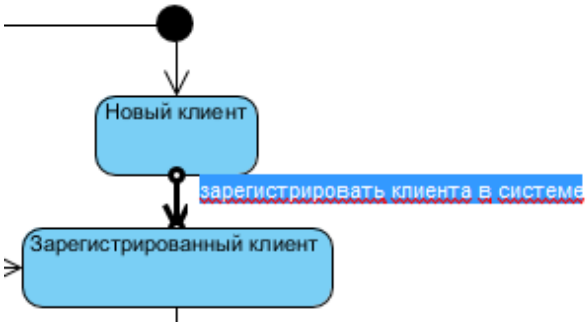


Рисунок 258 – Добавление описания перехода

Чтобы добавить переход разветвления или объединения выберите **Control Flow > Fork Node** или **Join Node** (рисунок 17). После добавления элемента, для того чтобы изменить ориентацию на горизонтальную, щелкните правой кнопкой мыши по элементу и в контекстном меню выберите **Orientation > Horizontal.**

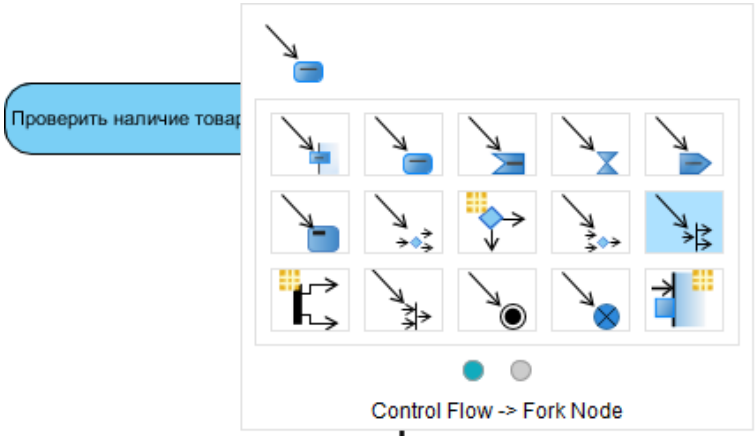


Рисунок 259 – Добавление перехода разветвления

Основываясь, на описанных выше принципах построения, постройте диаграммы деятельности для системы продажи товаров по каталогу в соответствии с рисунком 18.

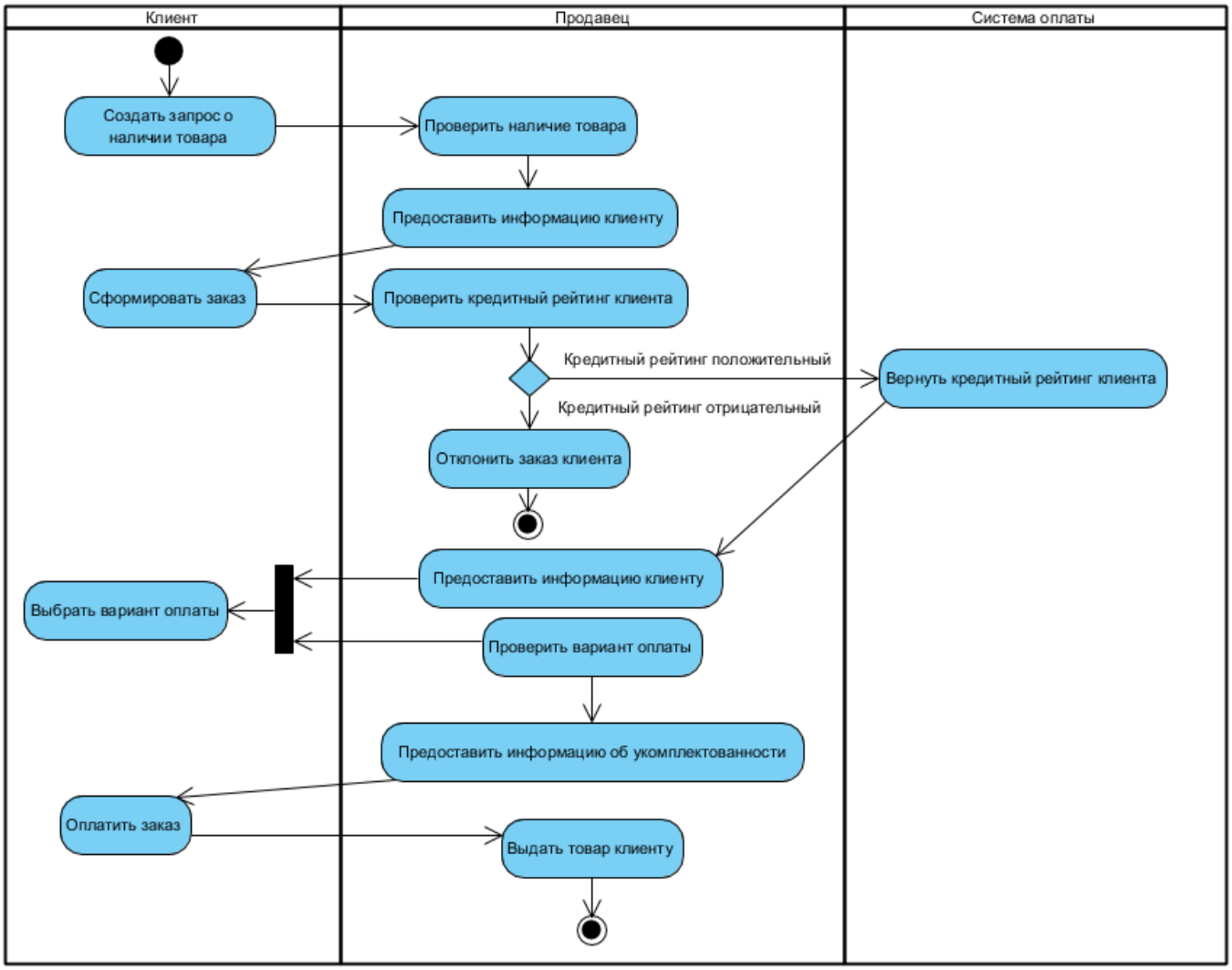


Рисунок 260 – Диаграмма деятельности системы продажи товаров по каталогу

5. Задание

Построить диаграмму деятельности в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;
10. «Таксопарк».

7. Контрольные вопросы

1. Дайте определение понятию «диаграмма деятельности».
2. Опишите назначение диаграммы деятельности.
3. Дайте определение понятиям «состояние деятельности» и «состояние действия». Графическое изображение состояния.
4. Приведите пример ветвления и параллельных потоков управления процессами на диаграмме деятельности.
5. Какие переходы используются на диаграмме деятельности?
6. Что представляет собой дорожка на диаграмме деятельности?
7. Как графически изображаются объекты на диаграмме деятельности?

Практическая работа №16

Разработка диаграмм последовательности с помощью Visual Paradigm for UML

1. Цель работы

Целью работы является изучение основ создания диаграмм последовательности на языке UML с помощью case-средства Visual Paradigm for UML CE, применение приобретенных навыков для построения объектно-ориентированных моделей определенной предметной области.

2. Задачи

Основными задачами практической работы являются:

– получить навыки создания диаграмм последовательности с помощью case-средства Visual Paradigm for UML CE.

3. Краткие теоретические сведения

Диаграмма последовательности (sequence diagram) – диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления.

На диаграмме последовательности неявно присутствует ось времени, что позволяет визуализировать временные отношения между передаваемыми сообщениями. С помощью диаграммы последовательности можно представить взаимодействие элементов модели как своеобразный временной график «жизни» всей совокупности объектов, связанных между собой для реализации варианта использования программной системы, достижения цели или выполнения задачи.

На диаграмме последовательности изображаются объекты, которые непосредственно участвуют во взаимодействии, при этом никакие статические связи с другими объектами не визуализируются. Диаграмма последовательности имеет как бы два измерения: одно – слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта; второе – вертикальная временная ось, направленная сверху вниз.

Каждый объект графически изображается в форме прямоугольника и располагается в верхней части своей линии жизни (рис. 1). Внутри прямоугольника записываются собственное имя объекта и имя класса, разделенные двоеточием, запись подчеркивается, что является признаком объекта, который, как указывалось ранее, представляет собой экземпляр класса.

Рисунок 261 – Графические элементы диаграммы последовательности

Объекты записываются так же, как и в диаграмме кооперации. **Анонимный объект** – такой объект, для которого отсутствует собственное имя, но указано имя класса. **Объект-сирота** – такой объект, для которого может отсутствовать имя класса, но указано собственное имя объекта. Роль классов в именах объектов на диаграммах последовательности, как правило, не указывается.

Крайним слева на диаграмме изображается объект-инициатор моделируемого процесса взаимодействия (объект a на рис. 1). Правее – другой объект, который непосредственно взаимодействует с первым, т.е. порядок расположения объектов на диаграмме последовательности определяется исключительно соображениями удобства визуализации их взаимодействия друг с другом.

Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом процесс взаимодействия объектов реализуется посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и образуют определенный порядок относительно времени своей инициализации. Другими словами, сообщения, расположенные на диаграмме последовательности выше, передаются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа «раньше-позже».

Линия жизни объекта (object lifeline) – вертикальная линия на диаграмме последовательности, которая представляет существование объекта в течение определенного периода времени. Линия жизни объекта изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей рабочей области диаграммы последовательности от самой верхней ее части до самой нижней («объект 1» и анонимный объект «Класса 2»).

Отдельные объекты, закончив выполнение своих операций, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML применяется специальный символ в форме латинской буквы «X». На рис. 2 этот символ используется для уничтожения анонимного объекта, образованного от «Класса 3». Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Рисунок 262 – Графическое изображение линий жизни и фокусов управления объектов

Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее производительность. В этом случае прямоугольник такого объекта изображается не в верхней части диаграммы последовательности, а в той, которая соответствует моменту создания объекта (анонимный объект, образованный от «Класса 3» на рис. 2). При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Объект создается со своей линией жизни, а, возможно, и с фокусом управления.

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия, или в состоянии пассивного ожидания сообщений от других объектов. Фокус управления – символ, применяемый для того, чтобы явно выделить подобную активность объектов на диаграммах последовательности.

Фокус управления (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 в форме вытянутого узкого прямоугольника («объект а» на рис. 1), верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона – окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни («объект а» на рис. 2), если на всем ее протяжении он активен.

Периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта фокусы управления изменяют свое изображение на линию жизни и наоборот (объект сирота «ob2» на рис. 2). Получить фокус управления может только объект, у которого в этот момент имеется линия жизни. Если же объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него может быть создан лишь экземпляр этого же класса, который, строго говоря, будет другим объектом.

В отдельных случаях объект может посылать сообщения самому себе, инициируя так называемые *рефлексивные* сообщения. Для этой цели служит специальное изображение (сообщение у «объекта а» на рис. 3). Такие сообщения изображаются в форме сообщения, начало и конец которого соприкасаются с линией жизни или фокусом управления одного и того же объекта. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента.

Если в результате рефлексивного сообщения создается новый подпроцесс или нить управления, то говорят о рекурсивном или вложенном фокусе управления. На диаграмме последовательности рекурсия обозначается небольшим прямоугольником, присоединенным к правой стороне фокуса управления того объекта, для которого изображается данное рекурсивное взаимодействие (анонимный объект «Класса 2» на рис. 3).

Рисунок 263 – Графическое изображение актера, рефлексивного сообщения и рекурсии

Сообщения на диаграмме последовательности имеют дополнительные семантические особенности. На диаграмме последовательности все сообщения упорядочены по времени своей передачи в моделируемой системе.

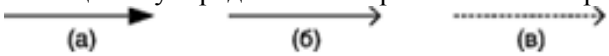


Рисунок 264 – Графическое изображение различных видов сообщений между объектами на диаграмме последовательности

Первая разновидность сообщения (рис. 4, а) наиболее распространена и используется для **вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления**. Начало этой стрелки, как правило, соприкасается с фокусом управления того объекта-клиента, который инициирует это сообщение. Конец стрелки соприкасается с линией жизни того объекта, который принимает это сообщение и выполняет в ответ определенные действия. При этом принимающий объект может получить фокус управления, становясь в этом случае активным. Передающий объект может потерять фокус управления или остаться активным.

Вторая разновидность сообщения (рис. 4, б) используется для обозначения **простого асинхронного сообщения**, которое передается в произвольный момент времени. Передача такого сообщения обычно не сопровождается получением фокуса управления объектом-получателем.

Третья разновидность сообщения (рис. 4, в) используется для **возврата из вызова процедуры**. Примером может служить простое сообщение о завершении вычислений без предоставления результата расчетов объекту-клиенту. В процедурных потоках управления эта стрелка может быть опущена, поскольку ее наличие неявно предполагается в конце активизации объекта. В то же время считается, что каждый вызов процедуры имеет свою пару – возврат вызова. Для непроцедурных потоков управления, включая параллельные и асинхронные сообщения, стрелка возврата должна указываться явным образом.

Каждое сообщение на диаграмме последовательности ассоциируется с определенной операцией, которая должна быть выполнена принявшим его объектом. При этом операция может иметь аргументы или параметры, значения которых влияют на получение различных результатов. Соответствующие параметры операции будет иметь и вызывающее это действие сообщение. Более того, значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвление или альтернативные пути основного потока управления.

Одна из **особенностей диаграммы последовательности** – возможность визуализировать простое *ветвление процесса.* Для изображения ветвления используются две или более стрелки, выходящие из одной точки фокуса управления объекта (объект «ob1» на рис. 5). При этом рядом с каждой из них должно быть явно указано соответствующее условие ветви в форме булевского выражения.

Количество ветвей может быть произвольным, но наличие ветвлений может существенно усложнить интерпретацию диаграммы последовательности. Предложение-условие должно быть явно указано для каждой ветви и записывается в форме обычного текста, псевдокода или выражения языка программирования. Это выражение всегда должно возвращать некоторое булевское выражение. Запись этих условий должна исключать одновременную передачу альтернативных сообщений по двум и более ветвям. В противном случае на диаграмме последовательности может возникнуть конфликт ветвления.

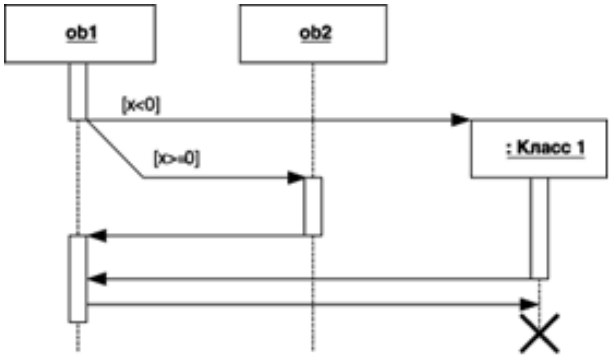


Рисунок 265 – Графическое изображение бинарного ветвления потока управления на диаграмме последовательности

Пример: объект «ob1» вызывает выполнение действий у одного из трех других объектов. Условием ветвления может служить сумма снимаемых клиентом средств со своего текущего счета. Если эта сумма превышает 1500$, то могут потребоваться дополнительные действия, связанные с созданием и последующим разрушением объекта «Класса 1». Если же сумма превышает 100$, но не превышает 1500$, то вызывается операция или процедура объекта «ob3». И, наконец, если сумма не превышает 100$, то вызывается операция или процедура объекта «ob2». При этом объекты «ob1», «ob2» и «ob3» постоянно существуют в системе. Последний объект создается от «Класса 1» только в том случае, если справедливо первое из альтернативных условий. В противном случае он может быть никогда не создан.

Объект «ob1» имеет постоянный фокус управления, а все остальные объекты – получают фокус управления только для выполнения ими соответствующих операций. На диаграммах последовательности при записи сообщений также могут использоваться стереотипы. Ниже представлена диаграмма последовательности для описанного выше случая ветвления, дополненная стереотипными значениями отдельных сообщений (рис. 6).

Сообщения могут иметь собственное имя, в качестве которого выступает имя операции, вызов которой инициируют эти сообщения у принимающего объекта. Тогда со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все равно изображаются. Построение диаграммы последовательности начинают с выделения из всей совокупности классов только тех, объекты которых участвуют во взаимодействии. После все объекты наносятся на диаграмму, с соблюдением порядка инициализации сообщений.

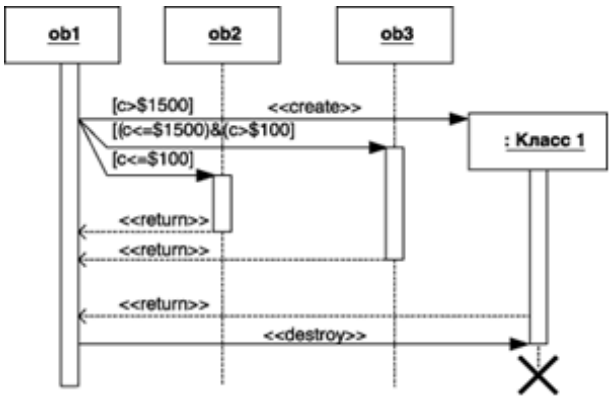


Рисунок 266 – Диаграмма последовательности со стереотипными значениями сообщений

Когда объекты визуализированы, можно приступать к спецификации сообщений, учитывая те операции, которые имеют классы соответствующих объектов, иногда используя их стереотипы.

Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение. Наиболее простые случаи ветвления процесса взаимодействия можно изобразить на одной диаграмме, а в более сложных случаях для моделирования каждой ветви управления может потребоваться отдельная диаграмма последовательности. Общим правилом является визуализация особенностей реализации каждого варианта использования на отдельной диаграмме последовательности.

Пример: Программное средство представляет среду для формирования отчетов по лекционному материалу. Пользователь может вводить свою информацию в отчеты, изменять параметры. После оформления отчетов пользователю предоставлена возможность сохранение отчета. Вывода его на печать и его архивацию.

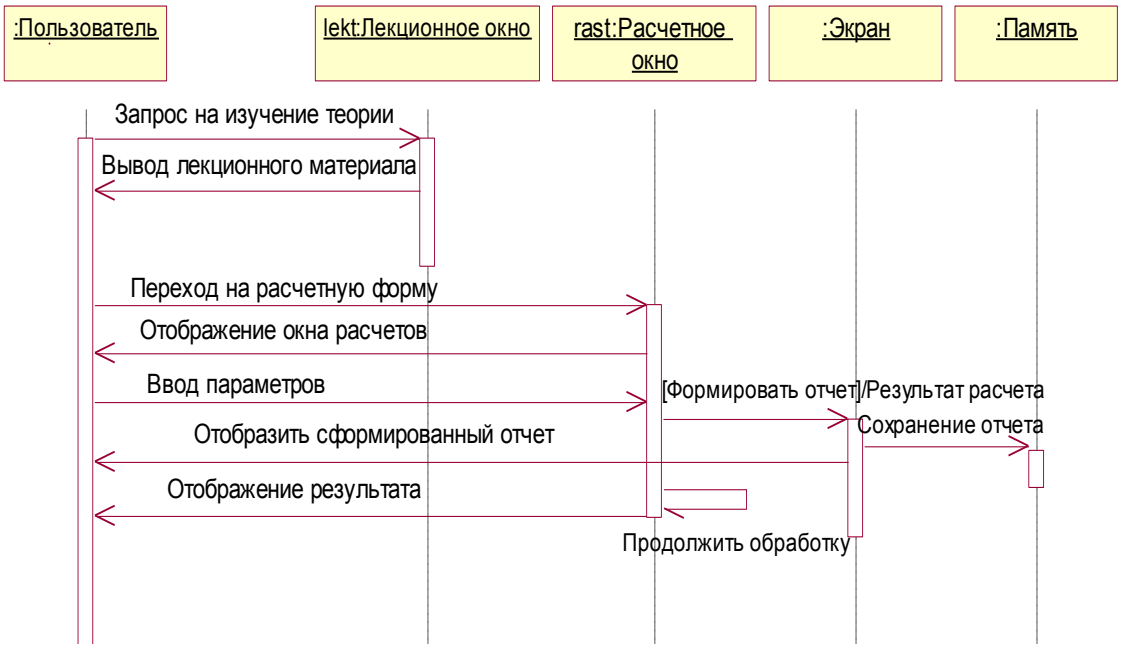


Рисунок 267 – Пример диаграммы последовательности

4. Методика выполнения

В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите Visual Paradigm for UML CE.
2. Откройте проект, содержащий диаграмму вариантов использования, построенную в практическом занятии №15.
3. Для добавления новой диаграммы достаточно в меню Diagram > New Diagram выбрать тип диаграммы Sequence Diagram и нажмите кнопку Next. Задать имя диаграммы SaleSystem SequenceDiagram и нажать OK.
4. В результате отобразится пустая рабочая область с элементами для построения диаграммы последовательности (рис. 8).

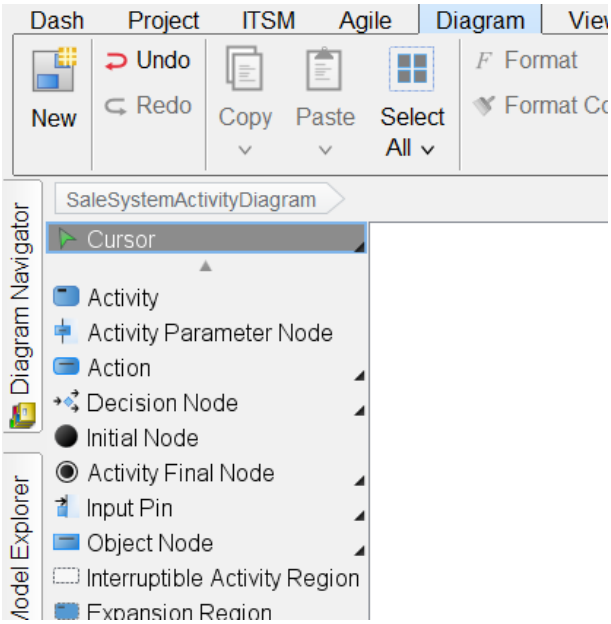


Рисунок 268 – Рабочая область для построения диаграммы последовательности

Далее будут описаны принципы построения диаграммы последовательности.

Для того чтобы создать актера, щелкните кнопкой мыши **Actor** на панели инструментов и затем щелкните на диаграмме (Рисунок 9).

Рисунок 269 – Создание Actor

Для создания линии жизни объекта нужно щелкните кнопкой мыши **LifeLine** на панели инструментов и затем щелкните на диаграмме. Другим более быстрым способом создания линии жизни объекта является способ создание через интерфейс ресурса (Рисунок 10). Нужно выбрать **Message** > **LifeLine** и перетащить курсор на пустое место на диаграмме, далее отпустить кнопку (Рисунок 11).

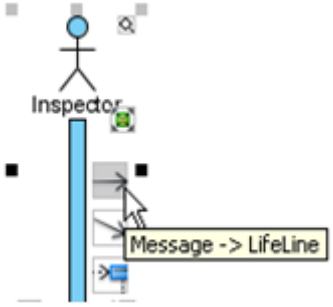


Рисунок 270 – Создание лини жизни объекта

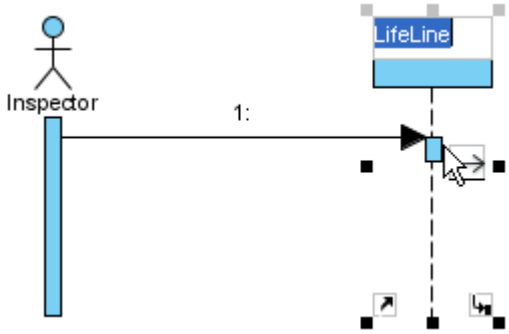


Рисунок 271 – Результат создания лини жизни объекта

Для того чтобы сделать диаграмму более читабельной можно воспользоваться инструментом **Sweeper**, позволяющий раздвигать элементы диаграммы последовательности (Рисунок 12).

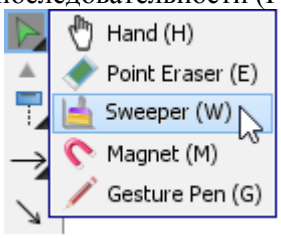


Рисунок 272 – Инструмент Sweeper

После выбора инструмента **Sweeper** перетащите его в нужном направлении, в этом же направлении будут перемещены и элементы диаграммы, находящиеся по другую сторону от **Sweeper** (Рисунок 13).

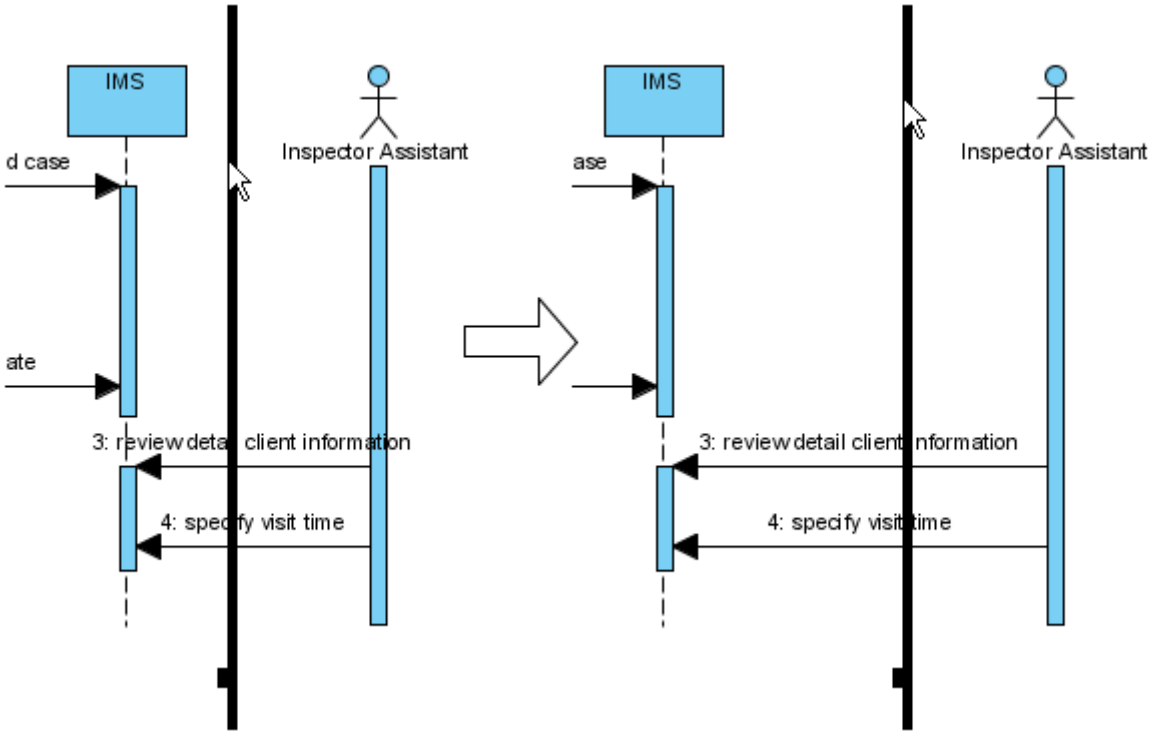


Рисунок 273 – Смещение элементов диаграммы последовательности вправо

Существует два способа нумерации сообщений **Diagram-based** и **Frame-based**.

При нумерации способом **Diagram-based** необходимо щелкнуть правой клавишей мыши на листе диаграммы и выбрать **Sequence Number,** а затем **Single Level** или **Nested Level** (Рисунок 14).

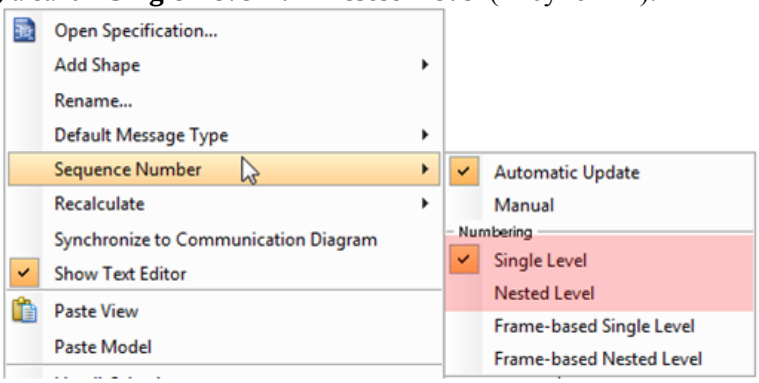


Рисунок 274 – Всплывающее меню при выборе нумерации сообщений Diagram-based способом

Если был выбран **Single Level**, то нумерация будет производится целыми числами, при выборе **Nested Level** способа – числами с десятичными знаками (Рисунок 15).

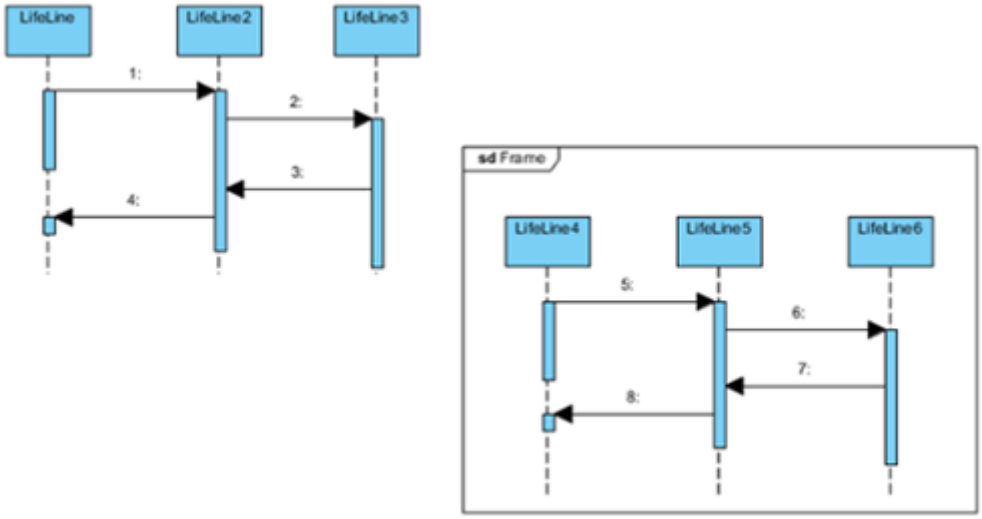


Рисунок 275 – Нумерация сообщений вариантом Diagram-based Single Level

При нумерации способом **Frame-based** необходимо щелкнуть правой клавишей мыши на листе диаграммы и выбрать **Sequence Number,** а затем **Frame-based Single Level** или **Frame-based Nested Level** (Рисунок 16).

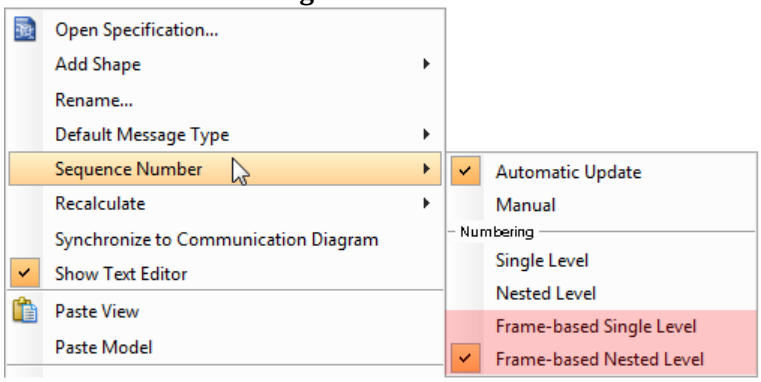


Рисунок 276 – Всплывающее меню при выборе нумерации сообщений Frame-based способом

При выборе нумерации сообщений способом **Frame-based** нумерация каждый раз начинается сначала в каждой независимой области (Рисунок 17).

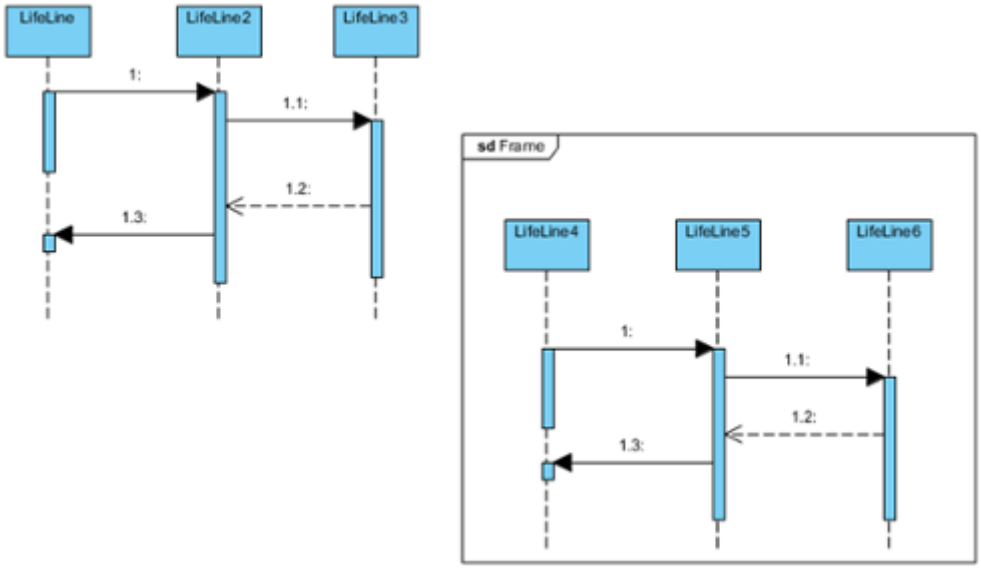


Рисунок 277 – Нумерация сообщений вариантом Frame-based Nested Level

Основываясь, на описанных выше принципах построения, постройте диаграммы последовательности для всех вариантов использования системы продажи товаров по каталогу в соответствии с рисунками 18-.

Для добавления диаграммы последовательности к варианту использования откройте диаграмму прецедентов, щелкните правой кнопкой мыши по варианту использования и в контекстном меню выберите **Sub Diagrams > New Diagram …** Затем в окне выбора диаграммы найдите **Sequence Diagram** и нажмите ОК.

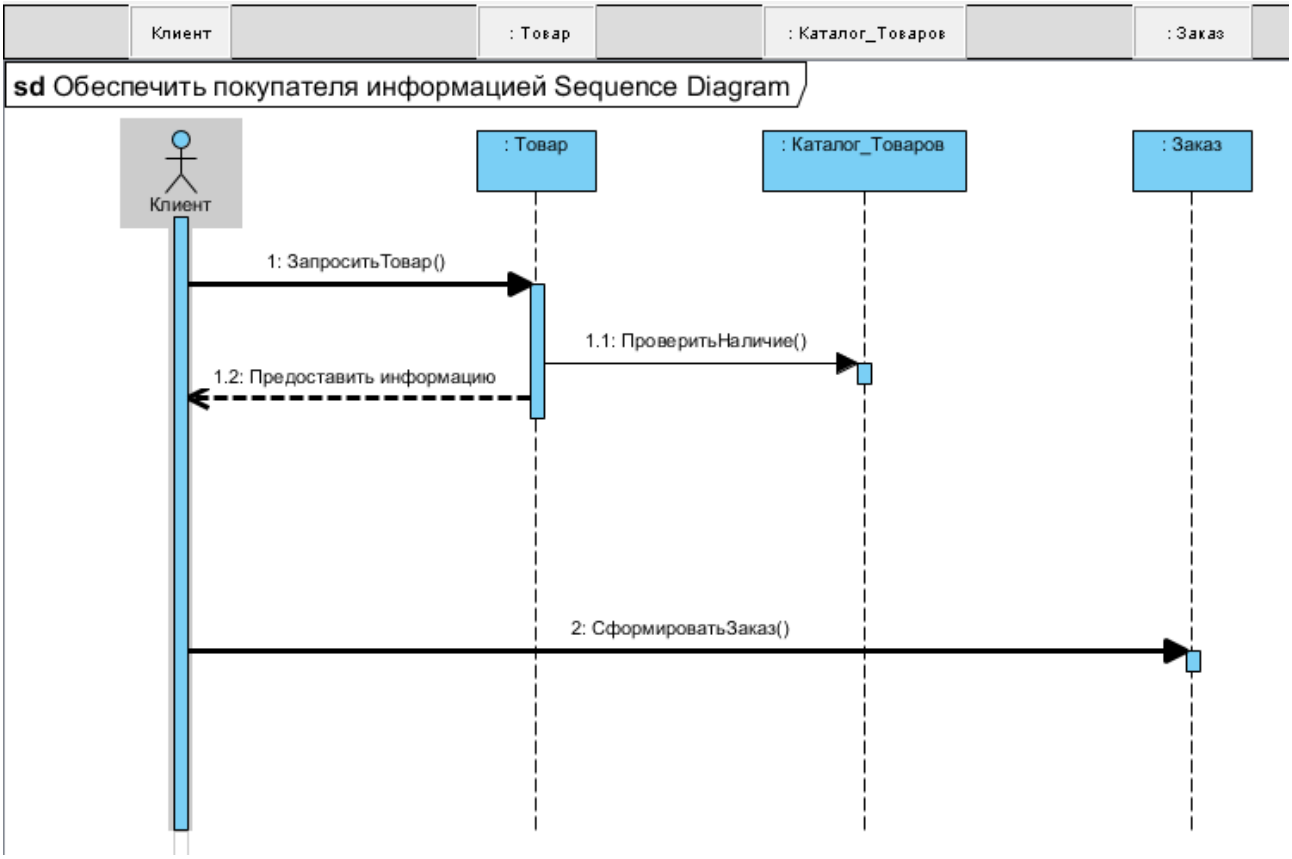


Рисунок 278 – Диаграмма последовательности для варианта использования «Обеспечить покупателя информацией»

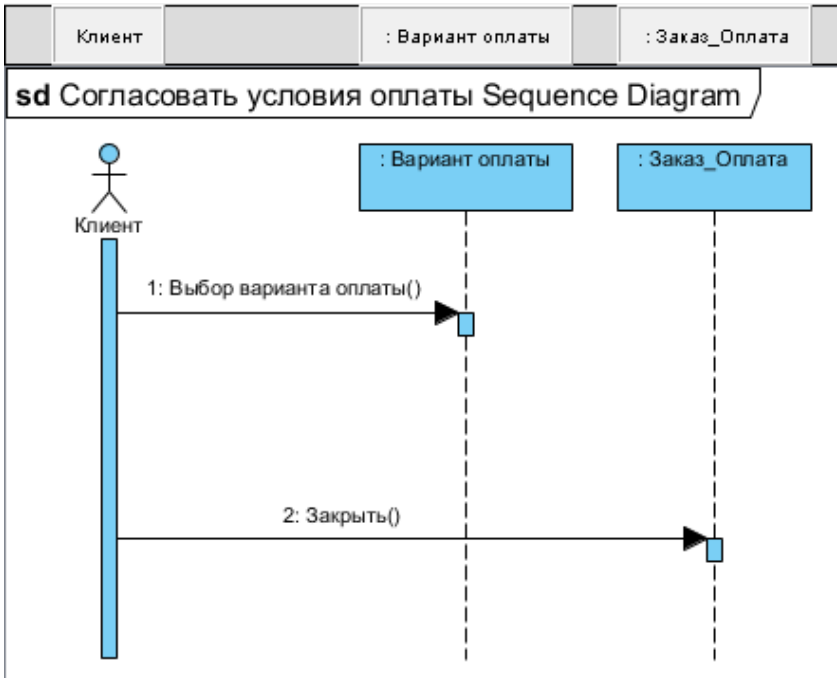


Рисунок 279 – Диаграмма последовательности для варианта использования «Согласовать условия оплаты»

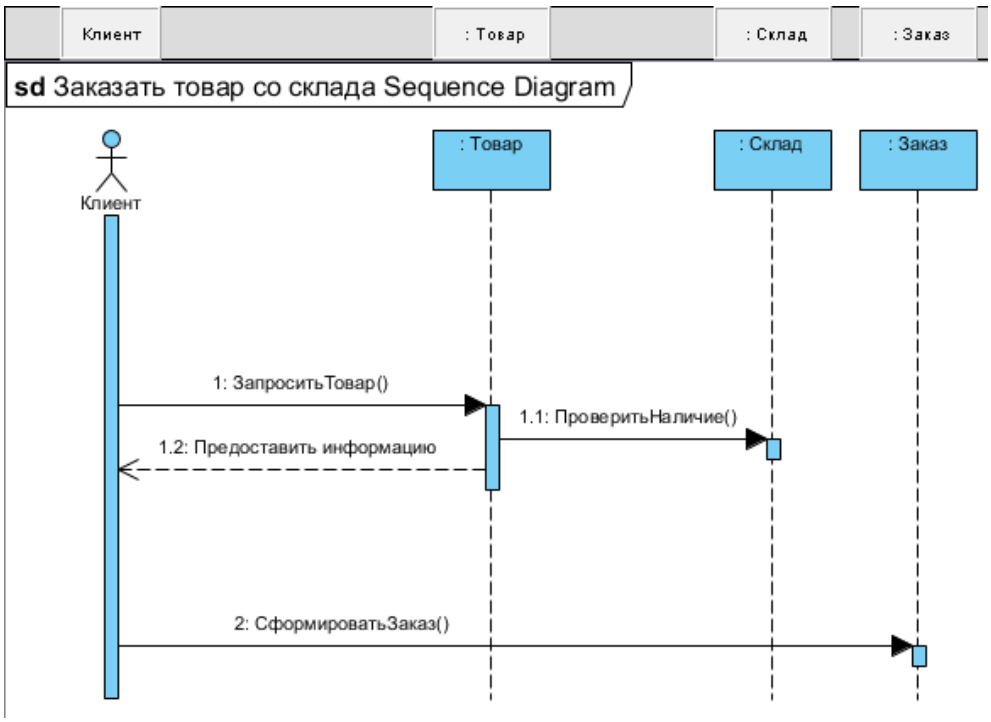


Рисунок 280 – Диаграмма последовательности для прецедента «Заказать товар со склада»

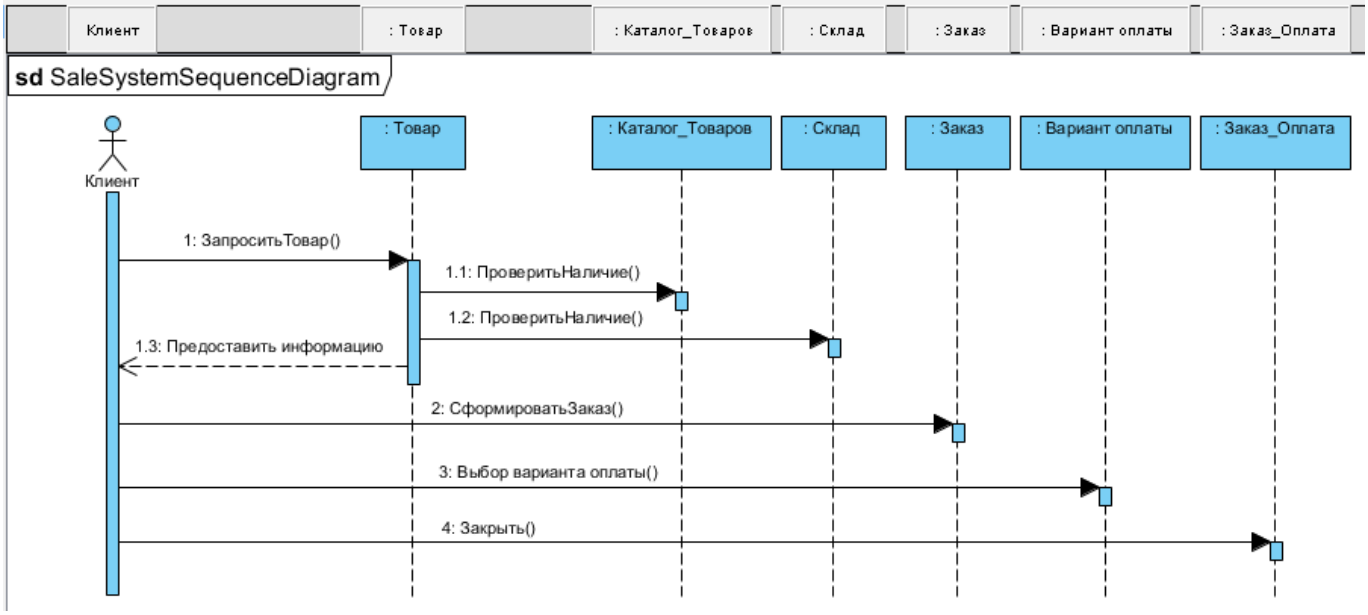


Рисунок 281 – Диаграмма последовательности для системы продажи товаров по каталогу

5. Задание

Построить диаграммы последовательности для всех вариантов использования (Практическая работа №12) и для всей системы в целом в соответствии с вариантом.

Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию.

6. Варианты

1. «Отдел кадров»;
2. «Агентство аренды»;
3. «Аптека»;
4. «Ателье»;
5. «Аэропорт»;
6. «Библиотека»;
7. «Кинотеатр»;
8. «Поликлиника»;
9. «Автосалон»;

10.«Таксопарк».

7. Контрольные вопросы

1. Дайте определение понятию «диаграмма последовательности».
2. Опишите назначение диаграммы последовательности.
3. В чем различие анонимного объекта и объекта-сироты?
4. Охарактеризуйте линию жизни объекта. Ее графическое изображение.
5. Для чего предназначен фокус управления? Его графическое изображение.
6. Дайте определение понятию «сообщение». Какие виды сообщений используются в диаграмме последовательности? Когда используется рефлексивное сообщение?
7. Расскажите про ветвления на диаграмме последовательности. Приведите пример ветвления.

Практическая работа №17

Разработка технического задания

1. Цель работы: знакомство со стандартами, регламентирующими жизненный цикл разработки ИС. Приобретение практических навыков при разработке программного документа Техническое задание.

2. Теоретические сведения

Для обеспечения потребностей по разработке программного обеспечения (ПО) ИС необходимо взаимосвязанное совершенствование технических решений, технологий проектирования и программирования, инструментальных средств, а также обучения специалистов.

Стандарты для процессов, инструментальных средств и данных, которые отражают лучшую практику и защищают от неблагоприятных последствий, играют определенную роль в обеспечении указанных потребностей, в частности, поддерживая доверие потребителя к продуктам, услугам и технологиям разработки ПО.

Использование стандартов при разработке ПО ИС позволит обеспечить:

- повышение надежности;
- повышение эффективности применения и снижение затрат в сфере сопровождения программных средств (ПС);
- увеличение коэффициента повторного использования ПС общего назначения;
- повышение объективности оценок качества ПС;
- создание условий для конкуренции разработчиков на внутреннем рынке ПС;
- обеспечение конкурентоспособности отечественных ПС на мировом рынке.

Стандарты комплекса ГОСТ 34 на создание и развитие автоматизированных систем (АС) – обобщенные, но воспринимаемые как весьма жесткие по структуре жизненного цикла (ЖЦ) и проектной документации. Наиболее популярными можно считать ГОСТ 34.601-90 **(**Автоматизированные системы. Стадии создания) и ГОСТ 34.602-89 (Техническое задание на создание АС). Введение единой, достаточно качественно определенной терминологии, наличие достаточно разумной классификации работ, документов, видов обеспечения и др. способствует более полной и качественной стыковке разных систем, что особенно важно в условиях, когда разрабатывается все больше сложных комплексных АС.

В последние годы в стране идет интенсивная работа по гармонизации государственных стандартов Российской Федерации с международными стандартами ISO в области создания нормативной базы управления жизненным циклом программных средств и информационных систем. В основе стандартизации - ГОСТ Р ИСО/МЭК 12207-99 "Информационная технология. Процессы жизненного цикла программных средств", который является аутентичным переводом международного стандарта ISO/IEC 12207: 1995-08-01.

Техническое задание (ТЗ) на АС - утвержденный в установленном порядке документ, определяющий цели, требования и основные исходные данные необходимые для разработки АС и содержащий предварительную оценку экономической эффективности.

ТЗ содержит основные технические требования, предъявляемые к изделию и исходные данные для разработки; в ТЗ указываются назначение объекта, область его применения, стадии разработки документации, её состав, сроки исполнения и т. д., а также особые требования, обусловленные спецификой самого объекта либо условиями его эксплуатации. Как правило, ТЗ составляют на основе анализа результатов, полученных в ходе предпроектного обследования.

Как инструмент коммуникации в связке общения заказчик-исполнитель, техническое задание позволяет:

А) Обеим сторонам:

- представить готовый продукт;
- выполнить проверку готового продукта по пунктам (приёмочное тестирование – проведение испытаний);
- уменьшить число ошибок, связанных с изменением требований в результате их неполноты или ошибочности (на всех стадиях и этапах создания, за исключением испытаний).

Б) Заказчику:

- осознать, что именно ему нужно, опираясь на существующие на данный момент технические возможности и свои ресурсы;
- требовать от исполнителя соответствия продукта всем условиям, оговорённым в ТЗ.

В) Исполнителю:

- правильно понять суть задачи, показать заказчику «технический облик» будущего программного изделия или АС;
- спланировать выполнение проекта и работать по намеченному плану;
- отказаться от выполнения работ, не указанных в ТЗ.

3. Задание к выполнению

В соответствии с вариантом задания, определяющим предметную область на основании ГОСТ 34.602-89, разработать документ Техническое задание на создание АИС.

4. Варианты

1. ИС «Отдел кадров»;
2. ИС «Агентство аренды»;
3. ИС «Аптека»;
4. ИС «Ателье»;
5. ИС «Аэропорт»;
6. ИС «Библиотека»;
7. ИС «Кинотеатр»;
8. ИС «Поликлиника»;
9. ИС «Автосалон»;
10. ИС «Таксопарк».

5. Контрольные вопросы

1. Этапы развития проекта ИС?
2. Понятие модели жизненного цикла ПО ИС?
3. Обзор моделей жизненного цикла, их недостатки и преимущества?
4. Обзор и особенности методологии *RAD*?
5. Обзор стандартов, регламентирующих ЖЦ ПО ИС?
6. Назначение, содержание и степень адаптивности стандарта ГОСТ 34.601-90?
7. Стадии и этапы создания АС в соответствии с ГОСТ 34.601-90?
8. Назначение, содержание стандарта ГОСТ 34.602-89?
9. Назначение, содержание и степень адаптивности стандарта ГОСТ Р ИСО/МЭК 12207?
10. Содержание одного из основных процессов ЖЦ ПО ИС по ГОСТ Р ИСО/МЭК 12207 – Разработка?
11. Содержание стандарта ISO/IEC 15288?
12. Назначение программного документа Техническое задание на создание АС. Порядок разработки, согласования и утверждения документа?
13. Состав и содержание документа ТЗ?