

ПРИЛОЖЕНИЕ А

Пособие по созданию UML диаграмм с помощью программы MS VISIO

1 Создание проекта

Для создания проекта необходимо запустить программу **Microsoft Office Visio** из группы установки системы в меню Программы Windows или двойным щелчком мыши по иконке рабочего стола Windows, если он установлен на рабочий стол. Окно программы после запуска представлено на рисунке А.1.

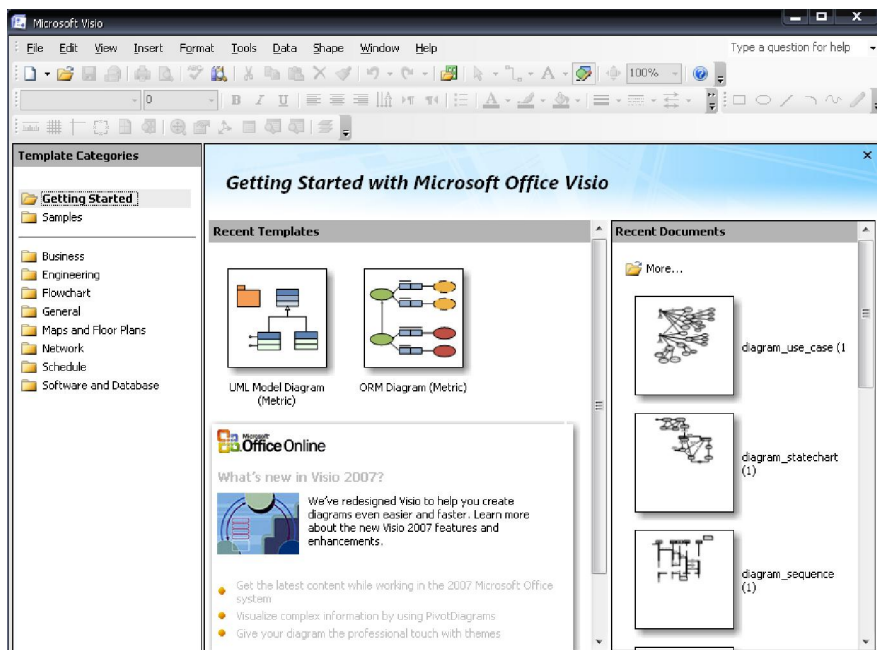


Рисунок А.1 — *Microsoft Office Visio 2007*

После запуска программы в меню **File** выбираем **New => Software and Database => UML Model Diagrams (Metric)**. Либо, в расположенном с левой стороны меню «**Template Categories**», выбираем пункт «**Software and Database**» и в раскрывшемся по центру списке двойным щелчком нажимаем на иконку «**UML Model Diagram**» (см. рис. А.2).

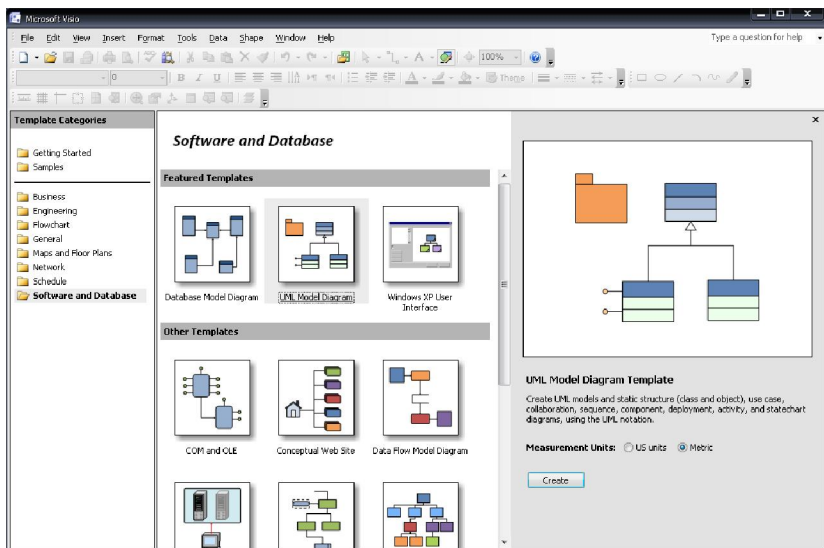


Рисунок А.2 — Создание нового проекта UML Model Diagrams

Проект моделирования после создания еще пустой. На левой стороне экрана вы увидите **Drawing Explorer** — проводник по модели (вы можете открыть его так же через **View => «Drawing Explorer Windows»**) (см. рис. А.3). Для добавления новых объектов в UML модели необходимо выбрать тип диаграммы. Для этого необходимо щелкнуть левой кнопкой мыши на соответствующем заголовке. А затем достаточно просто щелкнуть кнопкой мыши для выбора типа объекта.

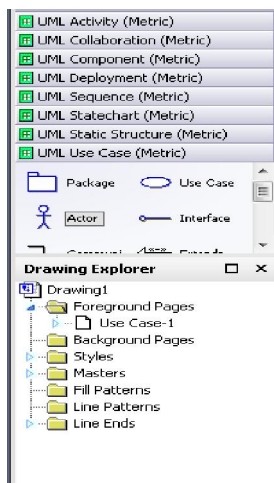


Рисунок А.3 – Окно Drawing Explorer

2 Создание диаграммы «Use Case»

UML диаграммы прецедентов (вариантов использования), иллюстрирует возможные сценарии в системе и внешние взаимодействующие с системой объекты (Actors), которые связаны с прецедентами использования (Use cases). Кроме того, могут быть показаны отношения между вариантами использования и приведены комментарии в случае необходимости.

Возможные отношения между вариантами использования:

- **отношения Include (Включения)** — используется, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования и необходимо, чтобы его описание копировалось в каждом из этих вариантов использования.

- **отношение Generalization (Обобщения)** — если есть один вариант использования, подобный другому, но намного шире (более глобальный).

- **отношение Extend (Расширения)** — расширяющий вариант использования может дополнять (делать более точным) поведение базового варианта использования, определив в базовом варианте использования точки расширения.

В качестве примера рассмотрим задачу: *проектирование телефонной службы приема заявок на покупку товаров*. Предполагаем, что заказчик этой системы — компания, которая внедряет новый сервис обслуживания клиентов по телефонным заявкам. Для этого клиенту необходимо зарегистрироваться, а затем по телефону сделать заказ необходимых товаров. Дальнейшие вопросы логистики (доставка и оплата) в систему не входят. Для этого компания организует телефонный центр, штат которого состоит из операторов, менеджера и группы технической поддержки.

Первым шагом по реализации этой задачи является уточнение требований и определение сценариев действия каждой роли в системе. Для этого и строится диаграмма Use Case. Диаграмма будет изображать внешних актеров, которые так или иначе взаимодействуют с системой и способы, которыми система может использоваться.

Actor — это типовые пользователи (менеджер, операторы и т. д.). Актер представляет собой некоторую роль, которую играет пользователь (или другая система) по отношению к системе.

Для построения диаграммы Use Case необходимо выполнить шаги:

1. Выбираем инструмент «**System Boundary**» (граница системы), нажимаем на него левой кнопкой мыши и, не отпуская её, перетаскиваем на поле (в рабочую область). Появится изображение в виде рамки (в дальнейшем рамку можно редактировать, уменьшать и увеличивать размер) (см. рис. А.4).

2. Как мы уже определились ранее — у нас выделены такие виды пользователей, как оператор, менеджер и техническая поддержка.

Из поля «**Shapes**» раздела «**UML Use Case**» выбираем актера (фигурка человечка) и перетягиваем его в рабочее поле за пределами рамки, изображенной ранее. Нажимаем дважды по нему — открывается окно свойств

«UML Actor Properties». В поле «Name» вписываем роль участника нашей системы, например, “оператор” и нажимаем кнопку ОК (см. рис. А.5).

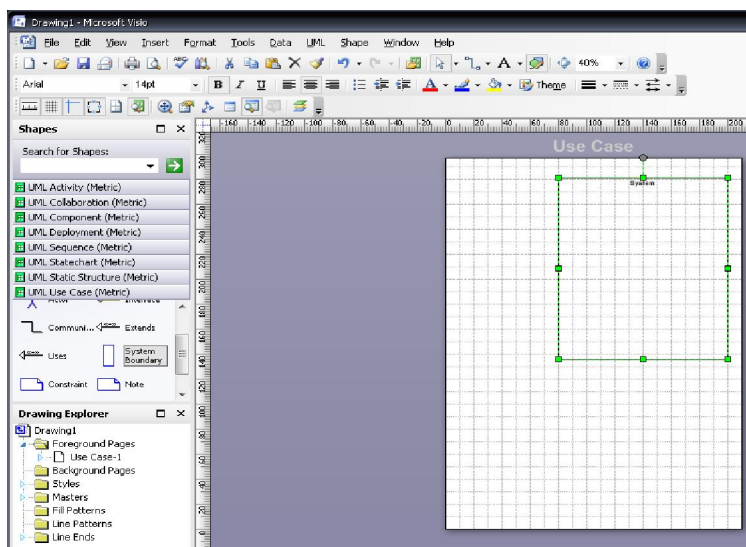


Рисунок А.4 — Вид инструмента «System Boundary» в рабочей области

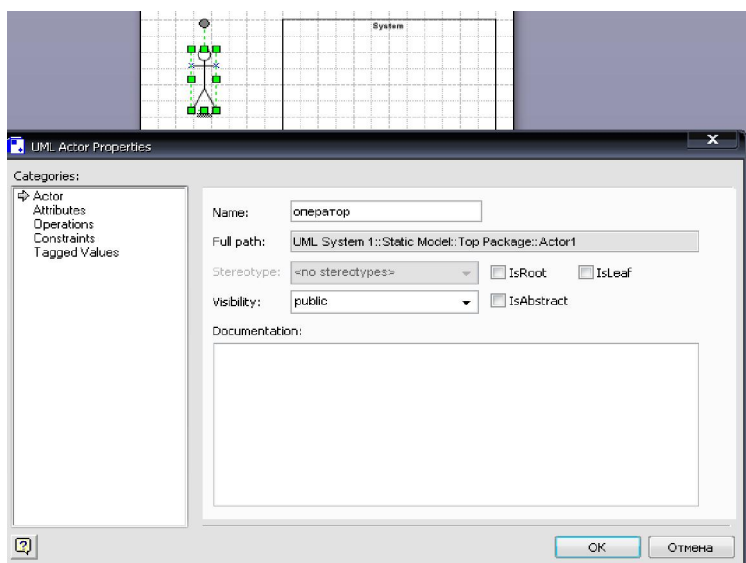


Рисунок А.5 — Окно свойств актера (для роли “оператор”)

Аналогичные действия выполняются для создания следующих ролей: **”менеджер”** и **”техническая поддержка”**.

3. Далее задается, как каждый из актеров взаимодействует с системой. Для этого используем элемент **«Use Case»** (Сценарий выполнения), несколько вариантов которого располагаем в изображенной рамке (внутри нашей системы). Для этого в списке на панели инструментов **«Shapes»** раздела **«UML Use Case»** выбираем элемент **«Use Case»** (элемент в виде овала) и перетаскиваем его в область нашей рамки (всем элементам, по необходимости, можно изменять размеры). Дважды нажимаем по нему, а затем в поле **Name** указываем названия действия, которое будет выполняться в системе, например, **“обработка запросов клиентов”** и нажимаем кнопку **ОК**. Один актер может иметь несколько вариантов использования системы, поэтому для каждого варианта добавляем отдельный элемент. Также один прецедент может выполняться несколькими ролями. К примеру, **“менеджер”** занимается просмотром текущих заявок, а также просмотром информации о рабочем времени операторов. Оператор обрабатывает запросы клиентов, занимается выбором справочной информации о клиентах и т. д. (см. рис. А.6).

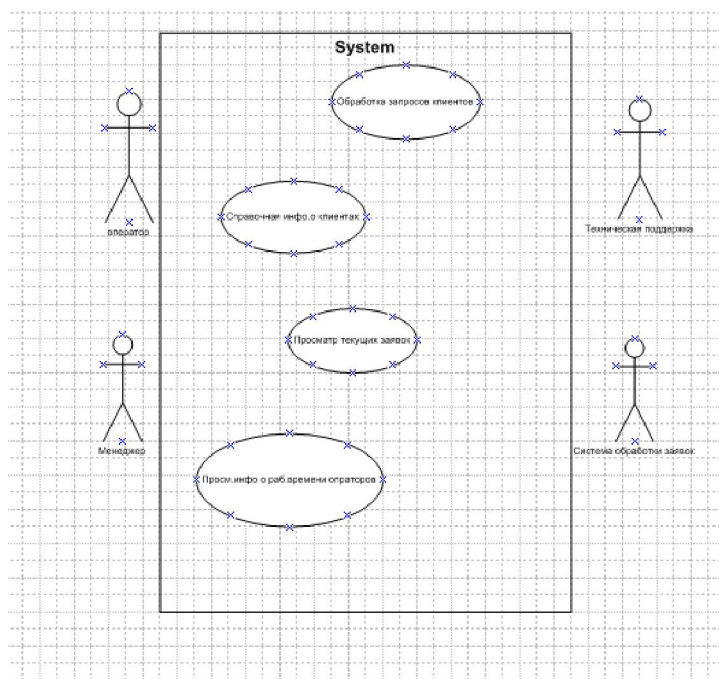


Рисунок А.6 — Изображение объектов и прецедентов использования

Теперь необходимо указать, как именно **Актор** осуществляет свою деятельность. Выбираем на панели инструментов элемент «Uses» (Использование) и перетягиваем в рабочую область, один конец соединяем с Актером, а другой – с областью использования (см. рис. А.7). Так мы создали связь между ролью (актером) и системой в виде варианта использования.

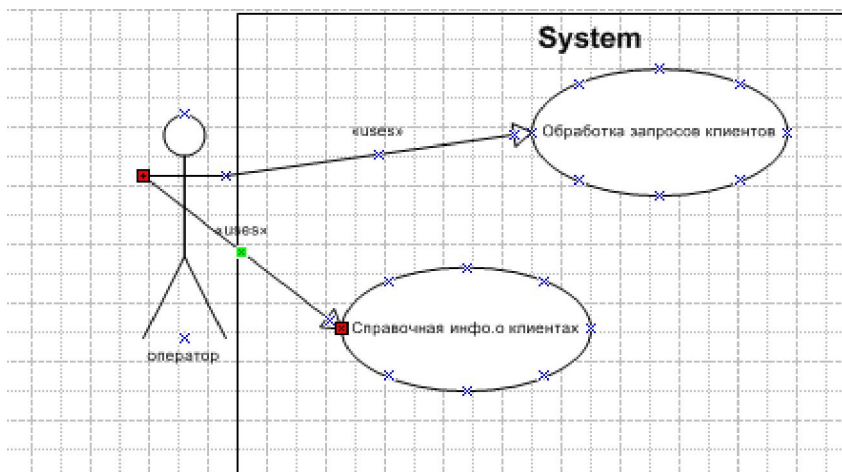


Рисунок А.7 — Изображение взаимодействия Актера с системой

Аналогично строим взаимодействия других ролей с системой, соединяя их с необходимыми прецедентами.

3 Создание диаграммы «Class»

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы (секции). В этих разделах указываются, соответственно: имя класса, атрибуты (переменные) и операции (методы).

Атрибуты (или **свойства**) класса записываются во второй сверху секции прямоугольника класса. В языке UML 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.

Полная форма атрибута:

<видимость> <имя>: <тип> <кратность> [= <значение по умолчанию> [{строка свойств}}]

Операции (или **методы**) класса записываются в третьей сверху секции прямоугольника. Операция (operation) представляет собой некоторый сервис, предоставляемый каждым экземпляром класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса. Обычно можно не показывать такие операции, которые просто манипулируют свойствами, поскольку они и так подразумеваются. Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам.

Полный синтаксис операций выглядит следующим образом:

<видимость> <имя> [(<список параметров>)] : <возвращаемый тип> [{<строка свойств>}]

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа «классификатор», которые связаны различными типами структурных отношений. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. На диаграммах классов отображаются также свойства классов, операции классов и ограничения, которые накладываются на связи между объектами.

Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы.

Для построения диаграммы Class необходимо выполнить шаги:

1. В меню «**Shapes**» (обычно слева) выбираем диаграмму «**UML Static Structure (Metric)**». На панели инструментов, во вкладке выбранной диаграммы, расположены специальные объекты — нотации (см. рис. А.8).

2. Выводим на экран элемент нашего класса. Для этого из списка элементов выбираем «**Class**» и перетаскиваем его в рабочую область. Дважды нажимаем по нему левой кнопкой мыши. В появившемся окне обозначаем имя класса, например, «**ATC_Agent**». Имя является обязательным элементов обозначения класса. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса. Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в первой верхней секции прямоугольника.

Созданный класс будет отвечать за работу непосредственно с АТС (устройство, которое занимается коммуникацией абонентов). В поле «**Visibility**» (Видимость) устанавливаем значение **public** (атрибут с этой областью видимости доступен (или виден) из любого другого класса пакета, в котором определена диаграмма). Доступны и другие типы видимости: **protected** (атрибут с этой областью видимости недоступен или невиден для

всех классов, за исключением подклассов данного класса) и **private** (атрибут с этой областью видимости недоступен или невиден для всех классов без исключения). В том же окне настройки класса переходим в другую категорию.

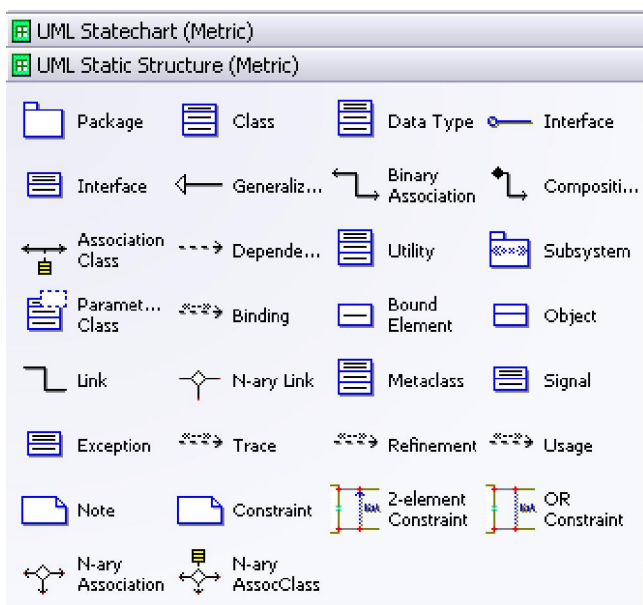


Рисунок А.8 — Элементы управления диаграммы «Static Structure»

Список категорий («Categories») изображен в этом окне слева. Выбираем категорию «Operation» (операции). Справа появится пустая таблица, которую необходимо заполнить.

Данный класс будет включать в себя такие функции как **Start** и **Stop**. Вписываем эти функции в таблицу (расположена в правой стороне окна) в колонку с названием **Operation**. Следующую колонку «**Return Type**» (Тип возврата) заполнять не будем, так как тип возврата нам пока не известен.

Если необходимо будет сгенерировать код, например, для команды **Stop**, тогда необходимо справа выбрать кнопку «**Methods**» (Методы) и в появившемся окне указать поля «**Operation name**» (имя класса), «**Language**» (язык программирования) и непосредственно «**Method body**» (тело метода). Чуть выше клавиши «**Method**» расположена клавиша «**Properties**» (Свойства), в окне которой можно также вносить настройки для соответствующей команды. Для указанных операций все настройки уже заранее известны, но в качестве примера, можно указать, например, тип возвращаемых данных: открываем графу «**Properties**» затем категорию «**Operation**», с правой стороны, в поле «**Return type**» устанавливаем тип возвращаемых данных `C#::int`. Тоже самое сделаем с операцией **Start**.

Завершаем заполнение нашего бланка нажатием на кнопку **ОК**. Таким образом, мы создали первый класс (см. рис. А.9)

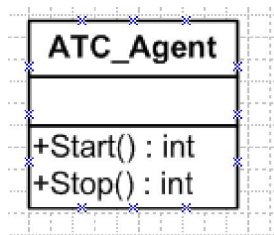


Рисунок А.9 — Нотация класса

Затем аналогично создаем класс **COperator**. В окне свойств для класса **COperator** выбираем категорию «**Attributes**» (атрибуты), и с правой стороны в таблице указываем необходимые значения. Например, укажем атрибут **ID**, для которого тип данных (**Type**) установим из списка в значение «**Top Package::COperator**» (см. рис. А.10). В категории **Operation** указываем соответствующую операцию **GetID** и тип возврата. В данном случае необходимо указать тип самого элемента — соответственно, устанавливаем «**Top Package::COperator**». Нажимаем на кнопку **ОК**.

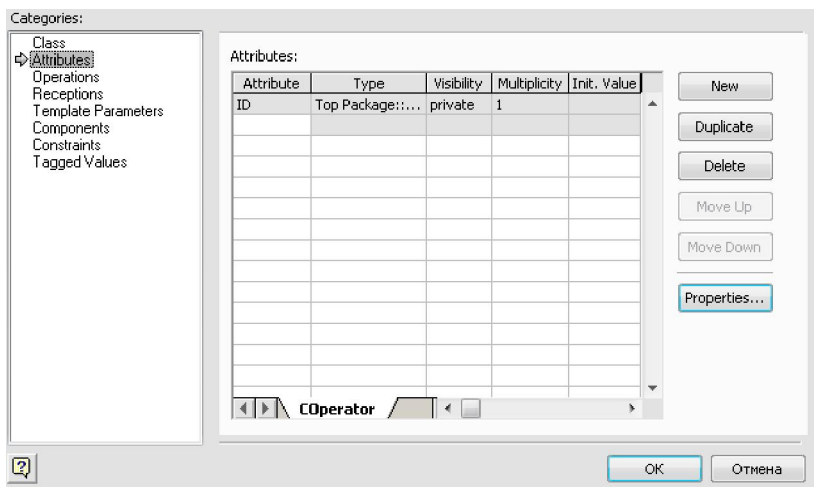


Рисунок А.10 — Окно свойств для класса COperator

Предположим, что имеется еще некий класс, который называется **SubOperator** и он описывает особенного «продвинутого» оператора, например, начальника операторов. Аналогично для него создаем класс, об атрибутах которого нам ничего не известно кроме его имени **CSubOperator**.

В данной задаче между классами **COperator** и **CSubOperator** существует связь, которая называется **ОБОБЩЕНИЕ**, она обозначается в виде стрелочки. Выбираем из списка элемент **«Generalization»** (Обобщение), перетягиваем его в рабочую область и соединяем указанные выше классы (см. рис. А.11.а).

По аналогии создаем остальные недостающие классы.

Создаем класс с именем **COperatorList** — это список классов, который управляет множеством операторов, т. е. добавляет в список, удаляет и т. д. Он включает в себя следующие операции: **Get** (захватить) и **Free** (освободить), тип возвращаемых данных указывать не будем.

Следующий класс, который создадим, будет называться **CServerNetCon**. Он обеспечивает соединение компьютеров через локальную сеть операторов. Имеет место следующие связи: **SetConnection** (Установить связь), **FreeConnection** (Освободить связь), **SendMessage** (Послать сообщение).

Последний класс, который может быть необходим, назовем **«CDispatcher»**. Он занимается синхронизацией всех остальных программных модулей на сервере. Он обладает функциями **IncomingCall** (Принимает звонки) и **ReceiveMessage** (Получает сообщения). Полная диаграмма описанных классов представлена на рисунке А.11.б.

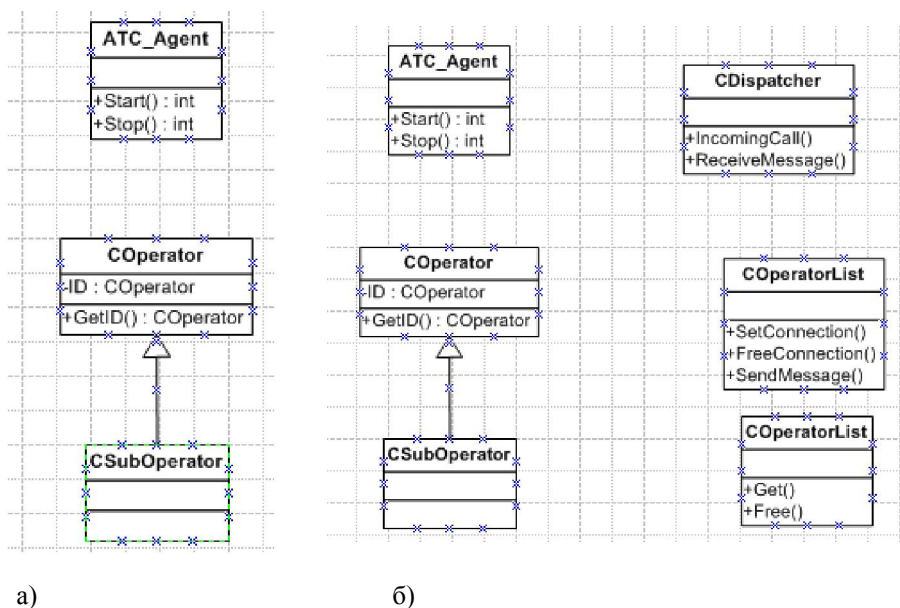


Рисунок А.11 — Диаграмма классов: а) изображение связи **«Generalization»** между классами **COperator** и **CSubOperator**; б) все необходимые классы для описанной задачи

3. Теперь необходимо указать зависимость между классами.

Базовыми связями в языке UML являются отношения:

- зависимости (dependency relationship),
- ассоциации (association relationship),
- обобщения (generalization relationship),
- реализации (realization relationship).

В первую очередь существует связь между элементами «**ATC_Agent**» и «**CDispatcher**». Так как в классе **ATC_Agent** **ATC** начинает работать (или останавливаться), и только после этого в класс **CDispatcher** поступают звонки. Поэтому выбираем в списке объектов элемент «**Link**» (связь) и соединяем указанные классы. В тоже время класс **CDispatcher** связан с классом **COperator**. В данном случае оператор занимается соответственно звонками. Эти классы так же соединяем связью «**Link**». Имеется связь между списком операторов **COperatorList** и **CDispatcher**, так как его задача-синхронизация всех имеющихся модулей. Далее связываем **CDispatcher** с **CServerNetCon**. В списке элементов есть связь под названием «**Composition**» (композиция). Данным взаимоотношением связаны элементы **COperator** и **COperatorList**. Полученная диаграмма представлена на рисунке А.12.

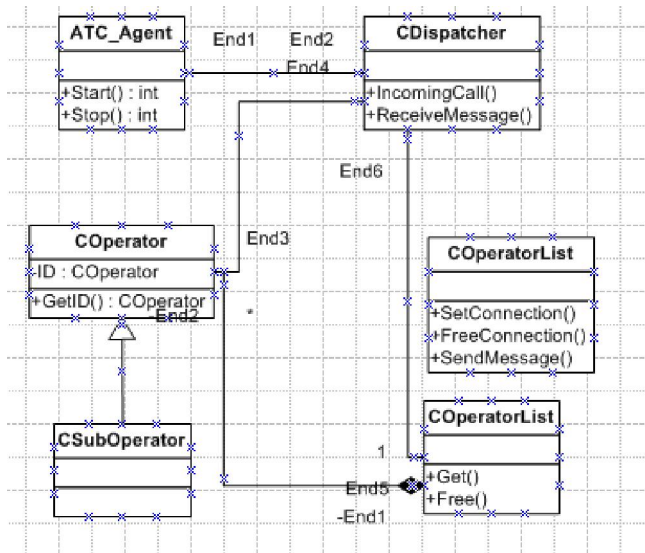


Рисунок А.12 — Итоговый вариант диаграммы классов

4. Создание «физических» диаграмм

4.1 Создание диаграммы развертывания (Deployment)

Диаграммы развертывания представляют физическое расположение системы, показывая, на каком физическом оборудовании запускается та или иная составляющая программного обеспечения.

Для новой диаграммы создаем новую страницу. Для этого нажимаем в рабочей области правой кнопкой мыши и в раскрывшемся меню выбираем **«Insert UML Diagram»**. Открывается окно, в котором необходимо указать название будущей диаграммы в поле **«Name»** и выбрать её тип (**«Deployment»**), затем нажимаем кнопку **OK**.

Слева, в окне инструментов **«Shapes»**, во вкладке диаграммы **«UML Deployment»**, расположены элементы данной диаграммы — это узлы, пакеты, компоненты, объекты и различные типы связи (см. рис. А.13).

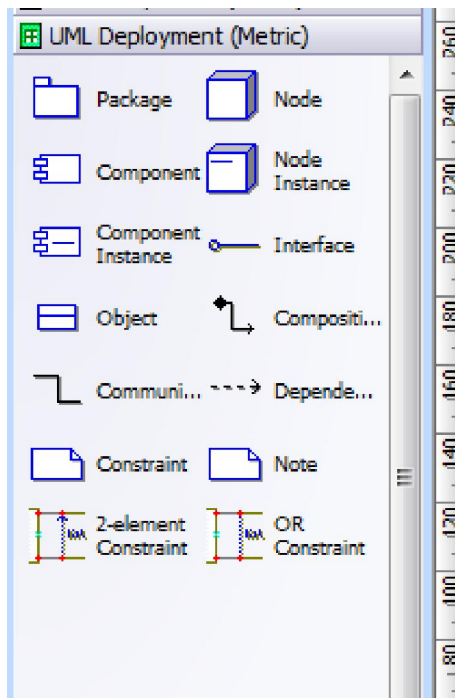


Рисунок А.13 — Элементы работы с диаграммой развертывания

Первыми на диаграмме отображаются основные элементы — физические узлы будущей системы. Для этого выбираем в панели

инструментов элемент **«Node»** и перетаскиваем его на рабочую область. За края элемента его можно растягивать (изменять размер). Открываем окно свойств узла двойным нажатием мыши по нему (либо один клик правой мыши на узле и в контекстном меню выбираем **«Properties»**). В поле **Name** вводим его имя (например, **«WebServer»**), нажимаем кнопку **OK**. Для каждого узла можно задать «метки» (свойства данного узла), которые отображаются в виде пары название-значение в фигурных скобках под названием узла. Чтобы добавить «метку» необходимо открыть окно свойств узла и перейти к вкладке **«Tagget value»**, выбрать соответствующий тэг и ввести его значение (например, *location=ServerRoom*). Если в списке нет подходящего названия, то можно добавить свой тэг с помощью кнопки **«New...»** в правой части окна. После заполнения всех тэгов нажимаем **OK**. Если «метки» не отображаются в узле, то нажимаем правой кнопкой мыши на узле и в контекстном меню выбираем **«Shape Display Options»**. В окне опций в блоке **«General options»** устанавливаем в активное состояние позицию **«Properties»**. Аналогичным образом создаем все узлы нашей системы (см. рис. А.14).

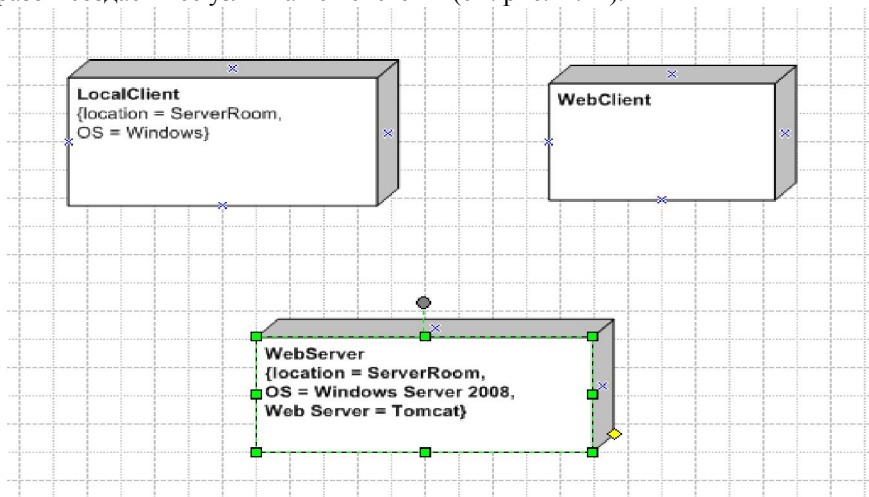


Рисунок А.14 — Рабочая область диаграммы развертывания с узлами

Узлы связаны друг с другом «информационными путями» (каналами связи, или коммуникациями). Для установки на диаграмме коммуникаций необходимо выбрать в панели инструментов элемент **«Communication»** и перетянуть в рабочую область. Затем концы соединить с двумя соответствующими узлами. Если это необходимо, то можно задать характер связи узлов, например, протоколы взаимодействия. Для этого открываем окно свойств элемента и в поле **«Name»** вводим имя (например, **«http/WWW»**), нажимаем кнопку **OK**. Таким же образом заполняем все информационные протоколы узлов (см. рис. А.15).

Зачастую в качестве узла подразумевается отдельное физическое устройство (например, компьютер), но иногда в качестве узлов описываются и специфическое программное обеспечение, которое может включать другое ПО (узел-среда выполнения). Для узла-среды выполнения можно указать его физический родительский узел (т. е. узел на котором он работает). Например, на рисунке A.15 узел «**EJB Container**» выступает в качестве узла-среды выполнения, а «**ApplicationServer**» — в качестве узла-устройства. На диаграмме развертывания такая принадлежность отмечается записыванием имени родительского узла через двоеточие после имени узла-процесса, либо отображается один узел внутри другого.

Узел может содержать внутри себя различные «артефакты», чаще всего это файлы или пакеты, или компоненты программ. Чтобы отобразить соответствующее программное обеспечение, работающее внутри узла, необходимо перетащить из панели инструментов соответствующий элемент внутрь узла. Так на рисунке A.15 показан пакет «**BusinessLogic**», который реализован внутри узла «**EJB Container**».

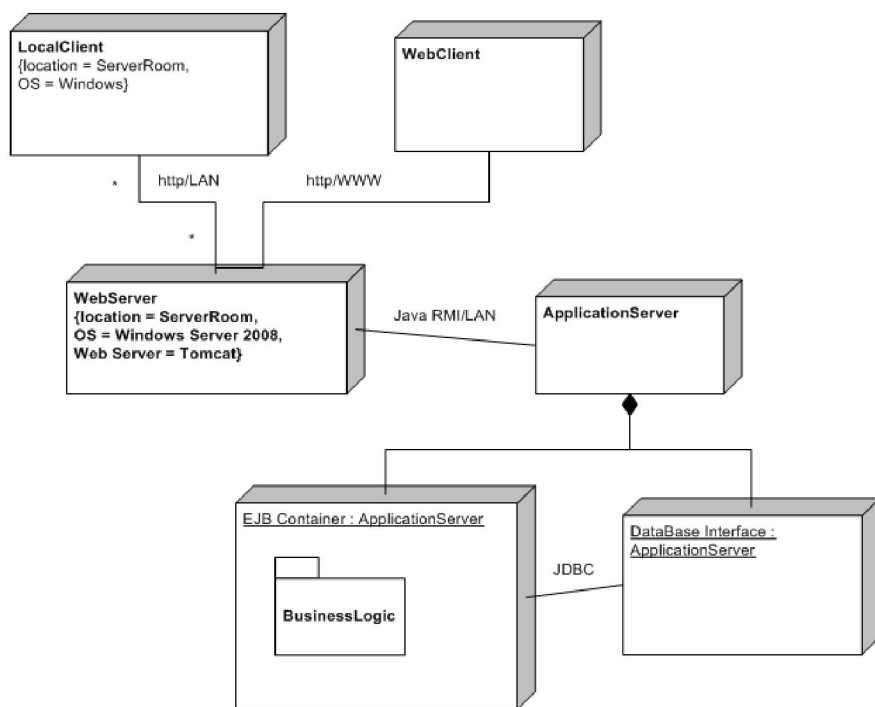


Рисунок A.15 — Диаграмма развертывания веб-ресурса

4.2 Создание диаграммы компонентов

Компоненты — это программные модули, которые обеспечивают независимое выполнение ряда задач. Компоненты можно считать «кирпичиками» будущей системы, из которых она и строится. Компоненты позволяют выполнить декомпозицию сложной системы на отдельные подзадачи, которые в свою очередь могут включать другие программные элементы (пакеты, классы).

Для диаграммы компонентов создаем новую страницу. Нажимаем по рабочей области правой кнопкой мыши и в контекстном меню выбираем **«Insert UML Diagram»**. В окне указываем название и тип необходимой диаграммы (**«Component»**), нажимаем кнопку **OK**.

Слева, на панели инструментов **«Shapes»**, во вкладке диаграммы **«UML Component»**, расположены специальные элементы для данной диаграммы. Основные из них — это независимые модули программного обеспечения (пакеты и компоненты), которые скрывают свою реализацию и взаимодействуют друг с другом через интерфейсы (см. рис. А.16).

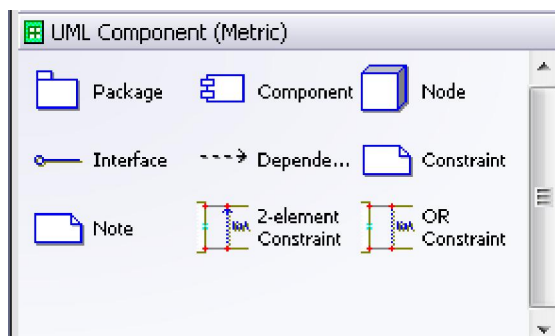


Рисунок А.16 — Элементы работы с диаграммой компонентов

В первую очередь из панели инструментов переносим элемент **«Component»** (компонент) в рабочую область. Двойным нажатием левой кнопки мыши открываем свойства данного компонента, в поле **«Name»** вводим его имя (например, **«ClientNetworkSupport»**), нажимаем кнопку **OK**. Это будет компонент, который обеспечивает поддержку сетевых команд.

Аналогично создаем второй компонент, который будет реализовывать пользовательский интерфейс рабочего места оператора. В качестве имени установим **«ClientGUI»** (см. рис. А.17).

Компоненты взаимодействуют через интерфейсы. На панели инструментов выбираем элемент **«Interface»** (интерфейс) и перемещаем его в рабочее пространство. Соединяем с компонентом **«ClientNetworkSupport»**. Двойным щелчком левой кнопки мыши открываем окно свойств данного

интерфейса и в поле «Name» указываем имя «**ICNSupportGUI**» (Interface Client Network Support GUI), нажимаем **OK** (см. рис. А.18).

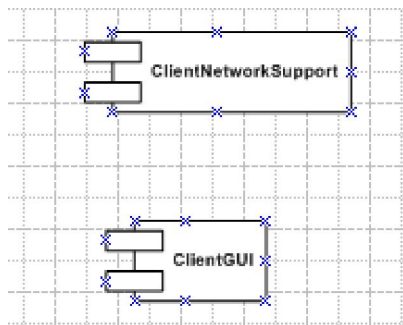


Рисунок А.17 — Компоненты «ClientGUI» и «ClientNetworkSupport»

Все имена интерфейсов должны начинаться с литеры «I».

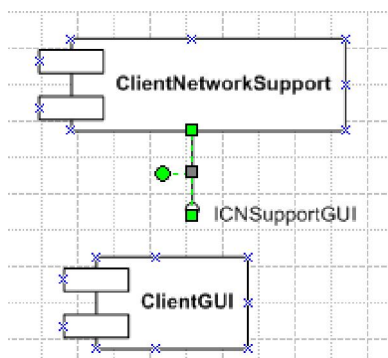


Рисунок А.18 — Интерфейс компонента ClientNetworkSupport

Аналогично клиентским компонентам, создаем в рабочей области компоненты сервера: **ServerNetworkSupport** (обеспечивает поддержку сетевых команд для сервера), **ServerBusinessLogic** (определяет бизнес-логику сервера, т. е. его внутреннее ядро), **RequestDB** (выполняет обращение к базам данным). Затем создаем интерфейс для компоненты, как было описано ранее, и устанавливаем зависимости компонентов (см. рис. А.19).

Компонентам можно указать тип — библиотека, документ, исполняемый файл, таблица и т. д. Для этого необходимо в окне свойств установить поле «**stereotype**».

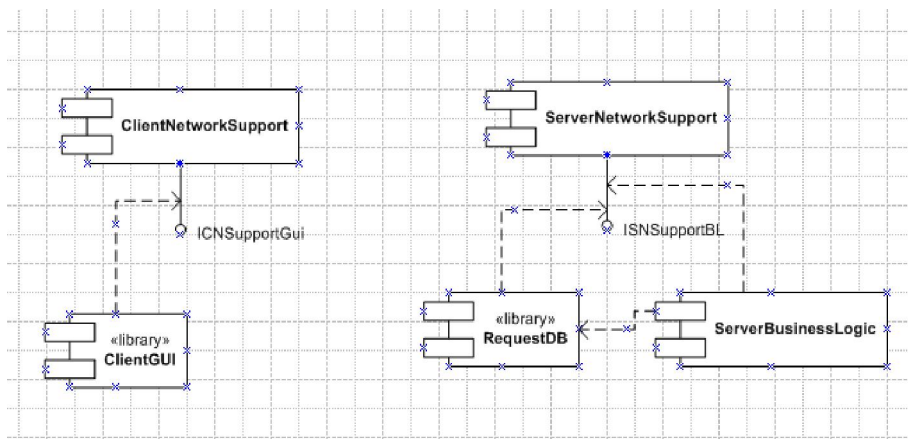


Рисунок А.19 — Диаграмма компонентов

Очень часто «физические» диаграммы (диаграммы развертывания и диаграммы компонентов) совмещают.

Построим совмещенную физическую диаграмму на основе нашей диаграммы компонентов. На новой странице в рабочее пространство перемещаем элемент «**Node**» (узел). Дважды нажимаем на нем и вызываем окно свойств, в поле «**Name**» вписываем имя «**Телефонный аппарат**», нажимаем **ОК**. Создаем еще два узла с именами «**АТС**» и «**Клиентский компьютер**». Затем из предыдущей диаграммы (см. рис. А.19) переносим всё, что относится к клиентскому компьютеру (это компоненты **ClientGUI** и **ClientNetworkSupport**) и вставляем в узел «**Клиентский компьютер**», предварительно увеличив его в размере (см. рис. А.20).

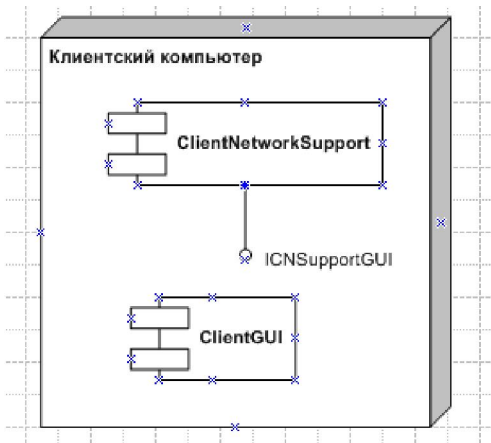


Рисунок А.20 — Узел «Клиентский компьютер» с компонентами

Соответственно, в узел «Сервер» необходимо перенести все оставшиеся компоненты.

В итоге получим расширенную диаграмму развертывания, которая включает взаимодействие компонентов (см. рис. А.21).

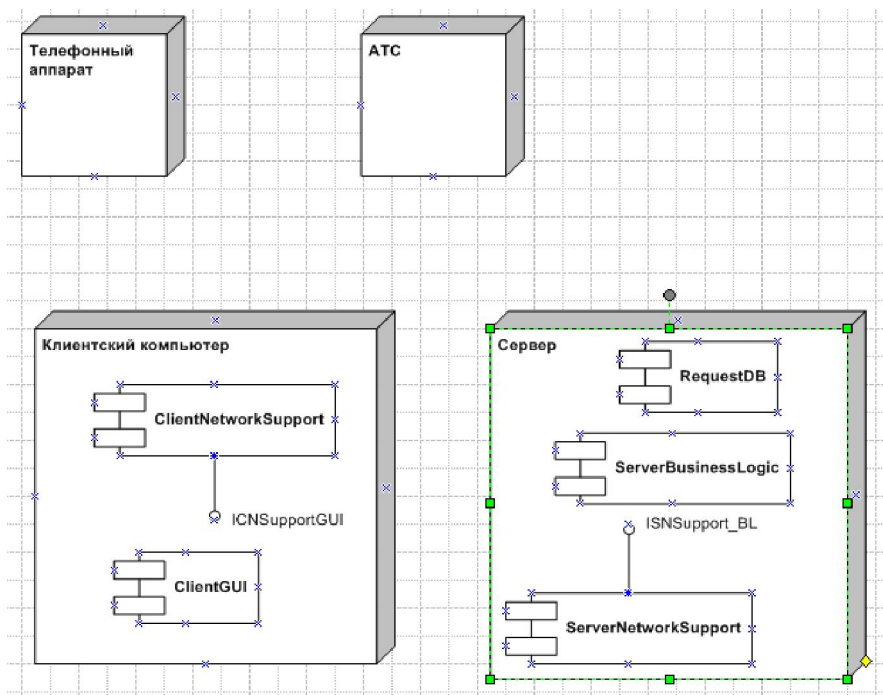


Рисунок А.21 — Расширенная диаграмма развертывания

ПРИЛОЖЕНИЕ Б

Пособие по созданию UML диаграмм в среде Microsoft Visual Studio 2010

1 Создание диаграммы развертывания

Рассмотрим создание диаграмм на проекте «Электронная библиотека».

Для построения диаграммы развертывания используем русскую версию *MS Visual Studio* и выполним следующие шаги:

1. В пункте меню «**Архитектура**» выбираем пункт меню «**Создать схему**».
2. Выбираем из списка «**Схема слоев**» (см. рис. Б.1).

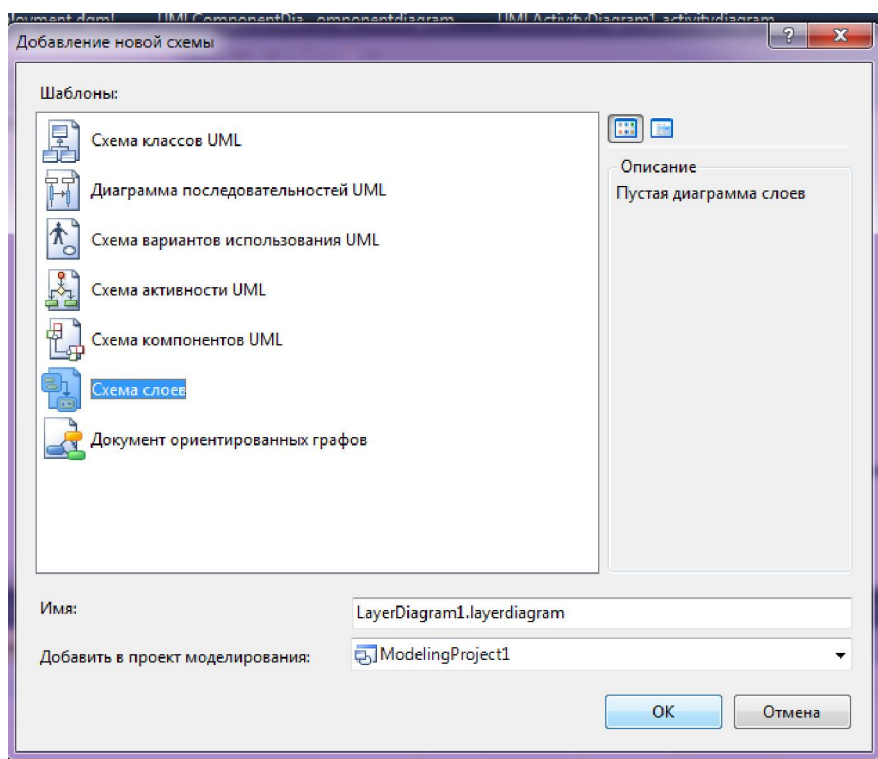


Рисунок Б.1 — Выбор типа диаграммы

3. Вызываем **Область элементов** (см. рис. Б.2).

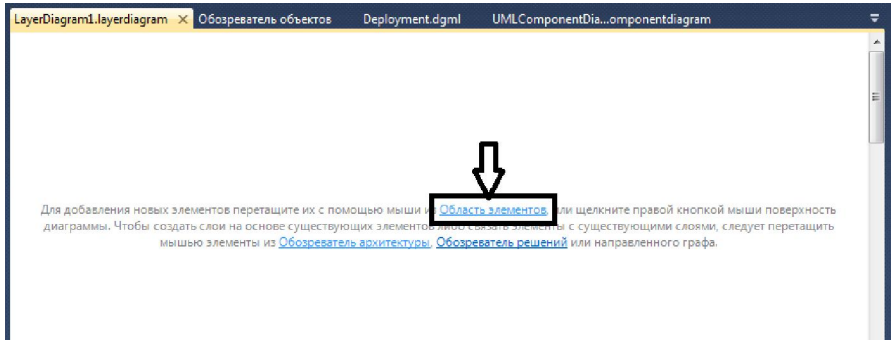


Рисунок Б.2 — Открываем рабочую область

4. Перетягиваем элемент «Слой» на рабочую область (см. рис. Б.3).

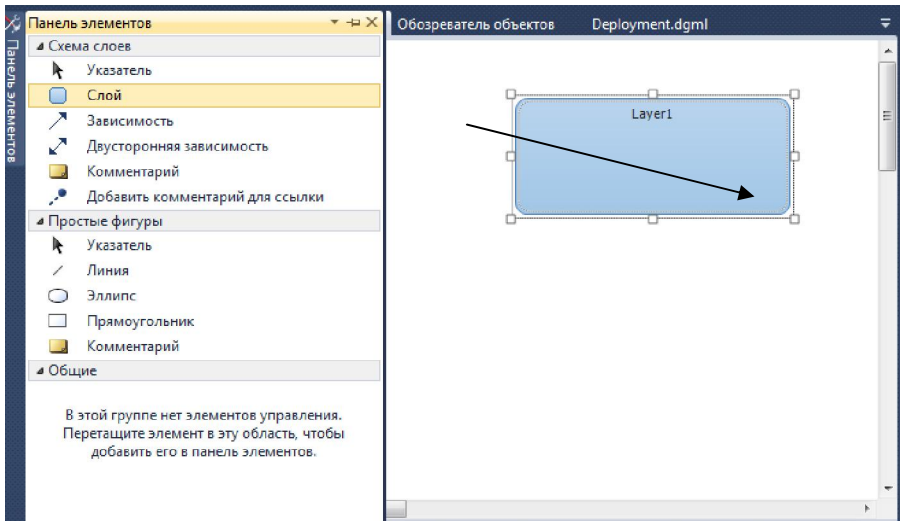


Рисунок Б.3 — Создаем узел

5. Переименовываем слой и добавляем новые слои (см. рис. Б.4).
6. Устанавливаем связь между слоями **АРМ** и **Сервер** при помощи элемента «**Двусторонняя зависимость**» (см. рис. Б.5).
7. Добавляем элемент «**Комментарий**» (см. рис. Б.6).
8. В свойстве комментария «**Прозрачный**» ставим значение **True**.
Диаграмма построена.

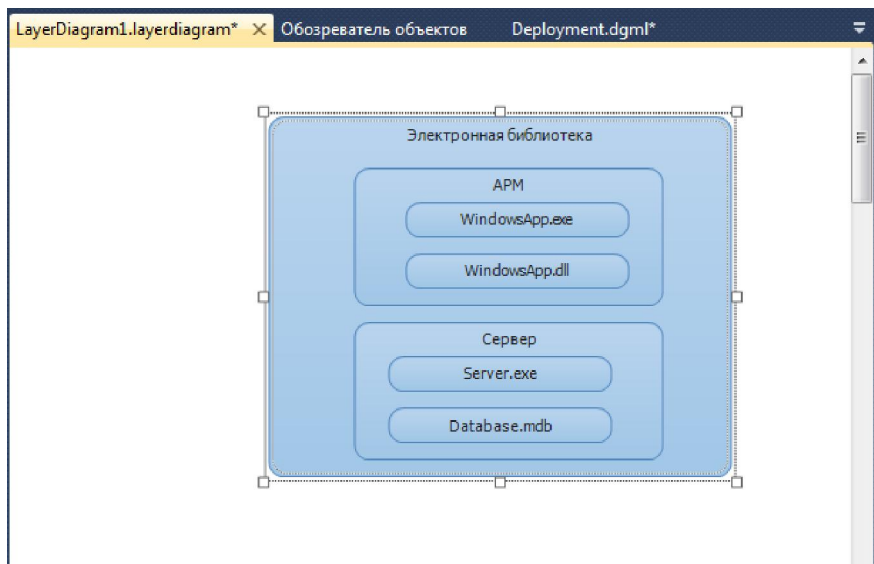


Рисунок Б.4 — Создаем артефакты в узле

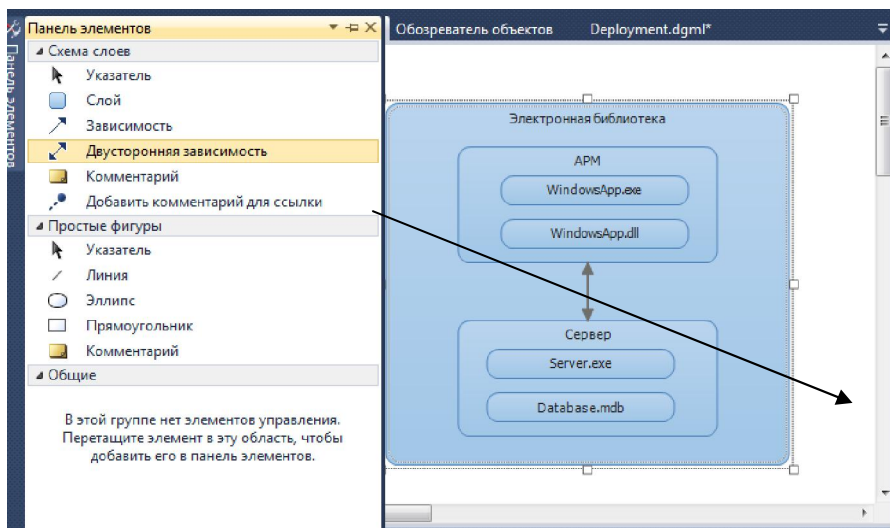


Рисунок Б.5 — Устанавливаем связь между узлами

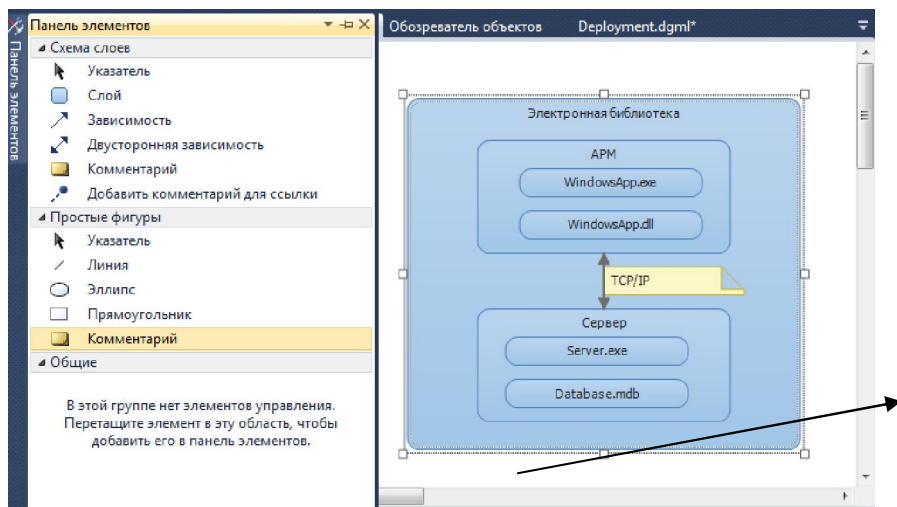


Рисунок Б.6 — Добавление комментария

2 Создание диаграммы деятельности

Для построения диаграммы деятельности используем английскую версию *MS Visual Studio* и выполним следующие шаги:

1. Добавляем в проект новый элемент «UML Activity Diagram»

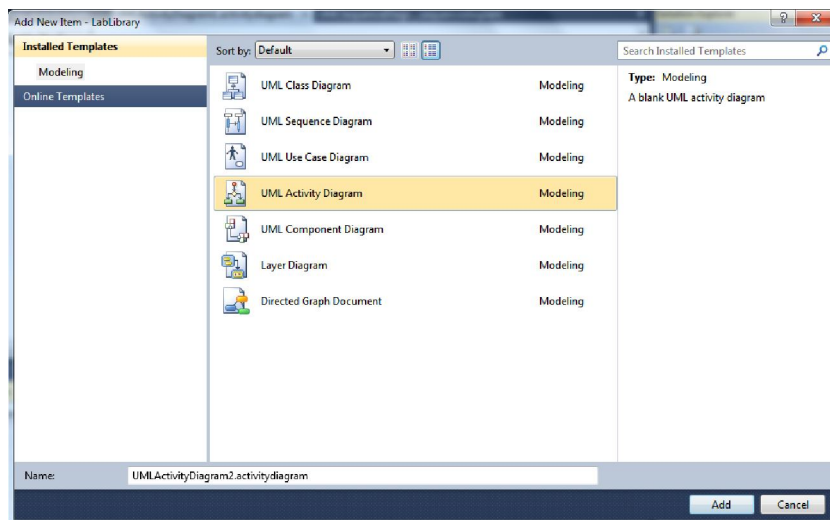



Рисунок Б.7 — Выбор диаграммы

2. Добавляем на диаграмму из панели инструментов элемент  **Initial Node** — начальную точку (см. рис. Б.8), обозначающую точку входа в программу.

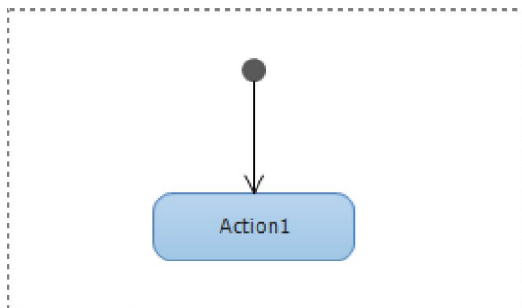






Рисунок Б.8 — Создание точки входа в программу и первого действия

3. На следующих шагах добавляем блоки, обозначающие действия программы из панели инструментов  **Action** (см. рис. Б.8). Соединяем точки перехода между действиями соединением коннекторами (соединительными линиями) из панели инструментов  **Connector** (см. рис. Б.8).

4. При необходимости добавляем блоки ветвления при наличии условий перехода к следующим действиям программы. При необходимости разделения процесса выполнения программы используем элементы разделения, а при объединении процессов в один — слияние.

5. Диаграмму можно сопровождать комментариями, добавляя следующий элемент из панели управления  **Comment** (см. рис. Б.9).

6. Завершается диаграмма деятельности добавлением конечной точки (конечное состояние) из панели инструмента элемента  **Activity Final Node** (см. рис. Б.9).

На рисунке Б.9 представлена законченная диаграмма деятельности, на которой указаны все блоки и элементы программы с комментариями и пояснениями.

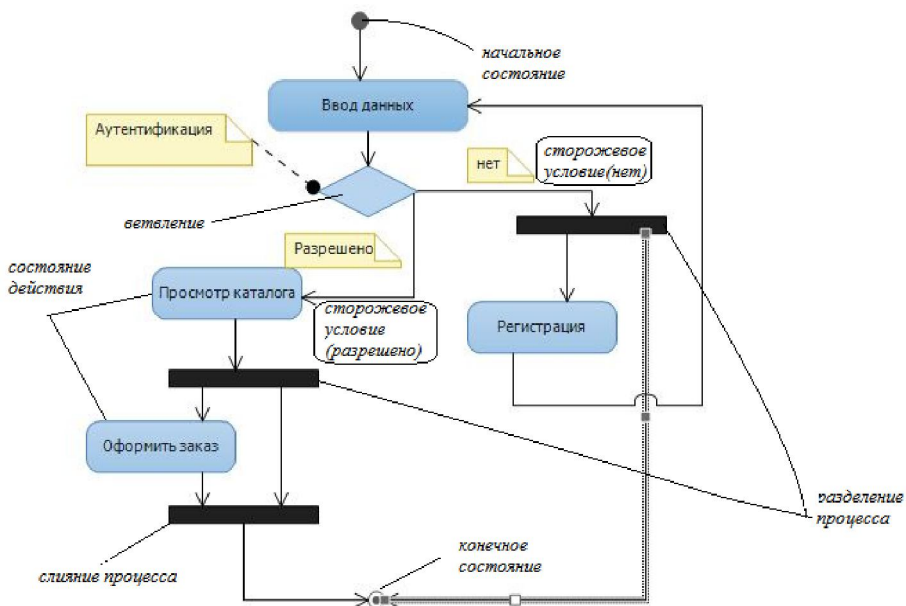


Рисунок Б.9 — Диаграмма деятельности

3 Создание диаграммы состояний

Для построения диаграммы состояний выполним следующие шаги:

1. Добавляем в проект новый элемент. Выбираем из списка — **UML Activity Diagram** (см. рис. Б.10).

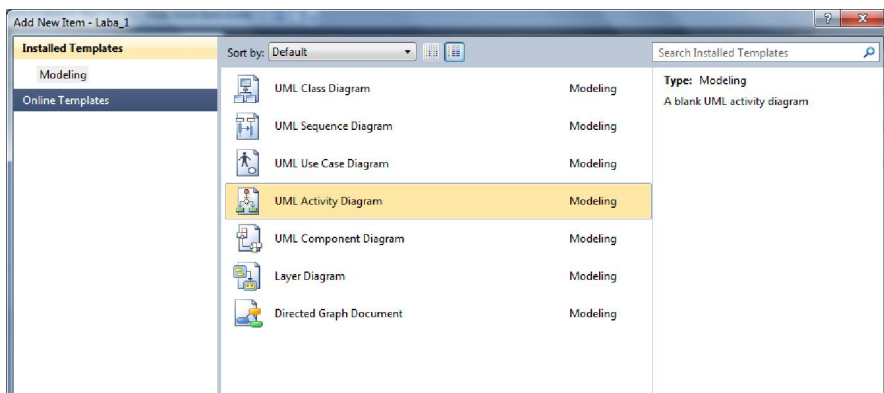


Рисунок Б.10 — Выбор диаграммы

2. Для отображения возможных состояний используем элемент, который перетаскиваем из вкладки «ToolBox» (см. рис. Б.11).

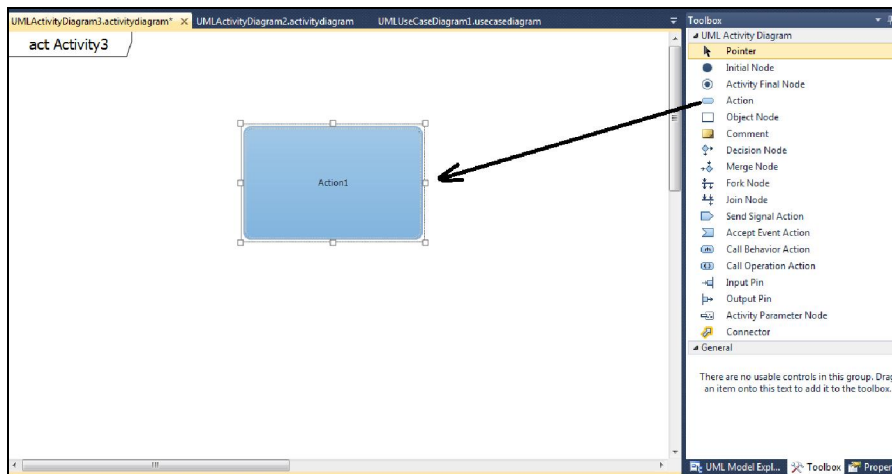


Рисунок Б.11 — Создание состояния

3. В верхней части созданного элемента необходимо указать состояние, а в нижней — деятельность, свойственную данному состоянию. Для этого в поле «Name» свойств элемента указываем названия состояния после горизонтальной линии, состоящую из символов «-» и название деятельности (см. рис. Б.12).

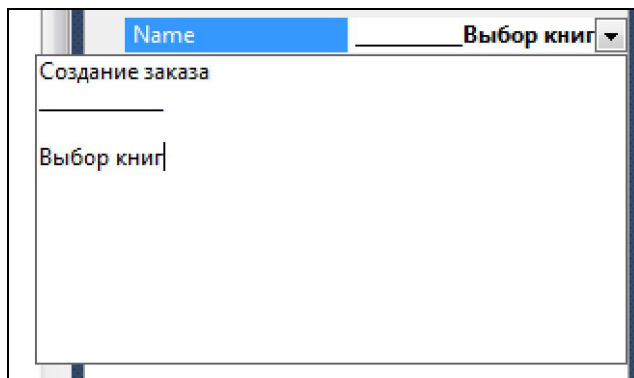


Рисунок Б.12 — Заполнение состояния

4. Соединять состояния необходимо с помощью элемента «**Connector**» (см. рис. Б.13). Переход из состояния в состояние описывается тремя характеристиками:

- 1) событие (когда начинать переход);
- 2) условие (возможен ли этот переход);
- 3) действие (что выполнять во время перехода).

Элементы разделяются знаком «/». Для описания перехода необходимо в свойстве «**Guard**» элемента «**Connector**» указать три описанные выше характеристики.

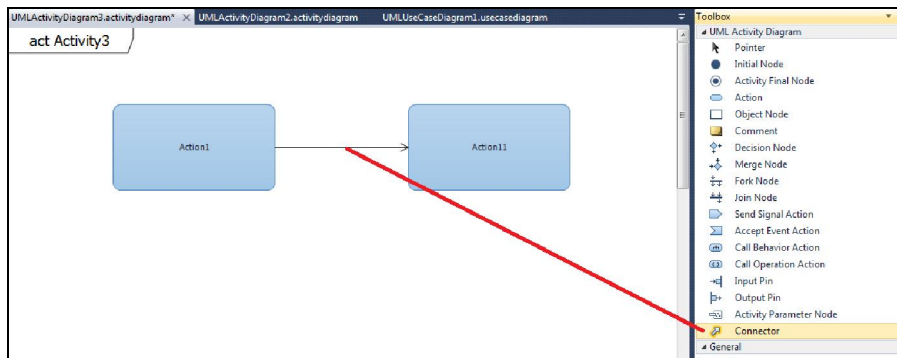


Рисунок Б.13 — Создание перехода между состояниями

5. Начальное положение описывается с помощью элемента «**Initial Node**». Конечное состояние — с помощью элемента «**Activity Final Node**» (см. рис. Б.14).

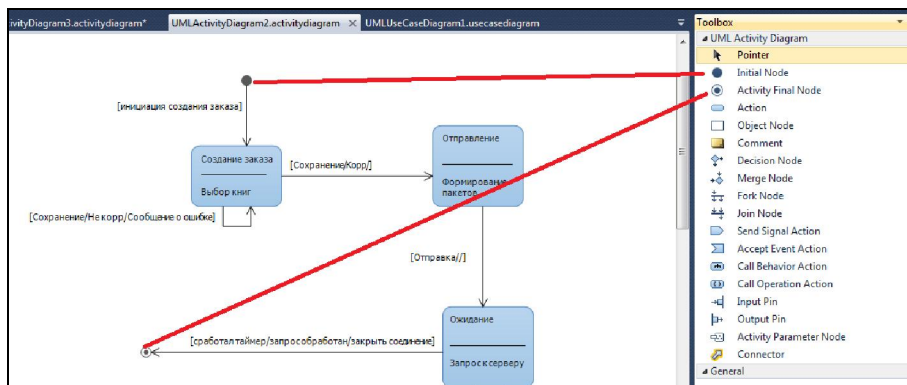


Рисунок Б.14 — Добавление старта и стока диаграммы

На рисунке Б.15 представлен пример диаграммы состояний.

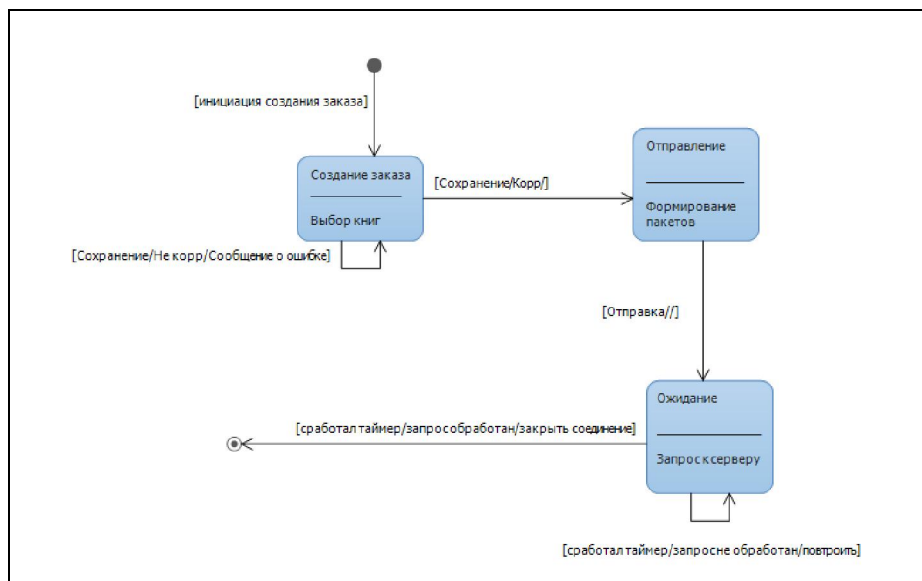


Рисунок Б.15 — Диаграмма состояний