минобрнауки россии

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**рАЗРАБОТКА ПРОГРАММНОГО**

**ОБЕСПЕЧЕНИЯ CРЕДСТВАМИ uml и aris**

Учебно-методическое пособие

Санкт-Петербург

СПбГЭТУ «ЛЭТИ»

2018

Авторы: С.С. Егоров, А.В. Соничев

Разработка программного обеспечения средствами UML и Aris: учебно-методическое пособие. СПбГЭТУ «ЛЭТИ», 2018. 86 с.

Содержит описание лабораторных работ, позволяющих освоить методы и средства проектирования и разработки программного обеспечения. В качестве инструментальной поддержки объектно-ориентированного метода используется Violet UML Editor; для анализа и моделирования бизнес-процессов - ARIS Express.

Предназначено для студентов, обучающихся по направлению 09.03.02 «Информационные системы и технологии».

Одобрено

Методической комиссией факультета компьютерных технологий и

информатики

в качестве учебно-методического пособия

© СПбГЭТУ «ЛЭТИ», 2018

**ЛАБОРАТОРНАЯ РАБОТА № 1**

**ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ**

**Цель работы**

Сформировать навыки разработки концептуальной модели программного обеспечения информационной системы.

**Задание**

Разработать диаграмму вариантов использования (диаграмму прецедентов, use case diagram) программного обеспечения выбранной системы (приложение 2) и написать комментарии к ней.

**Порядок выполнения**

***1. Описание диаграммы***

Диаграмма вариантов использования служит описанием сервисов или функциональных требований к моделируемой системе и является первым этапом процесса объектно-ориентированного анализа и проектирования. Диаграммой является граф специального вида, в котором выделяются два типа вершин, – актеры и варианты использования; дуги определяют некоторые фиксированные типы взаимосвязей между ними.

***1.1. Актеры***

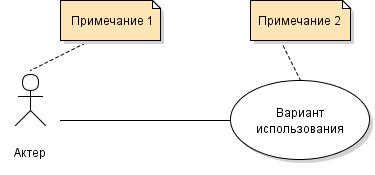
*Актером* (*actor*) - называется любая сущность, взаимодействующая с системой извне. Актеробозначается на диаграмме «человечком», под которым с заглавной буквы записывается его имя (рис. 1.1). Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. Актеры обозначают роли, которые могут играть в процессе взаимодействия с проектируемой системой. Примерами актеров являются, например, клиент банка, банковский служащий, продавец магазина, менеджер отдела продаж, пассажир авиарейса, водитель автомобиля, администратор гостиницы, сотовый телефон и другие сущности, имеющие отношение к концептуальной модели соответствующей предметной области.

***1.2. Варианты использования***

*Вариант использования* (*use case*) определяет сервисы, которые система предоставляет актеру, и задает некоторый набор действий, совершаемый системой при диалоге с актером (как реализовано взаимодействие актеров с системой, не указывается). Вариант использованияобозначается на диаграмме эллипсом, внутри которого содержится его краткое название или имя в форме глагола с пояснительными словами (рис. 1.1). Вариант использования определяет законченный аспект или фрагмент поведения системы, а так же любой ее части, которая обладает собственным поведением. Множество вариантов использования должно определять все возможные стороны ожидаемого поведения системы. Примерами вариантов использования могут являться, например, проверка состояния текущего счета клиента, оформление заказа на покупку товара, получение дополнительной информации о кредитоспособности клиента, отображение графической формы на экране монитора и другие действия.

***1.3. Примечания***

*Примечания* (notes) предназначены для включения в модель произвольной текстовой информации, относящейся к общему контексту системы. Графически примечания обозначаются прямоугольником с "загнутым" верхним правым уголком (рис. 1.1). Внутри прямоугольника содержится текст примечания. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия.

****

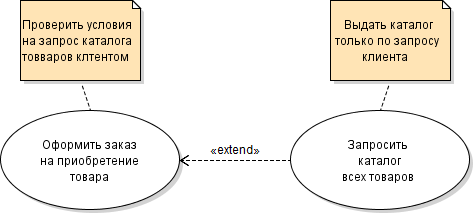
*Рис. 1.1*. Символы диаграмм вариантов использования

***1.4. Отношения***

*Отношения*на диаграмме вариантов использования описывают взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов использования. При этом выделяется несколько стандартных видов таких отношений

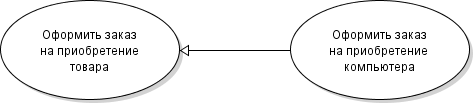
**А. *Отношение взаимодействия*** (interaction relationship) описывает значимую связь между двумя и более сущностями и является наиболее общим видом отношения. Применительно к диаграммам вариантов использования оно служит для обозначения специфической роли актера в отдельном варианте использования и 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 взаимодействия обозначается сплошной линией между актером и вариантом использования (рис. 1.1).

**Б. *Отношение расширения*** (*extend relationship*) определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров, и отмечает тот факт, что один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования. Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой, 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. 1.2.



*Рис. 1.2*. Пример графического изображения отношения расширения между вариантами использования

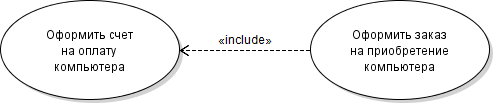
**В. *Отношение обобщения*** (*generalization relationship)* указывает на то, что некоторый вариант использования A может быть обобщен до варианта использования B. В этом случае вариант A будет являться специализацией варианта B. При этом B называется предком или родителем по отношению A, а вариант A - потомком по отношению к варианту использования B. При этом потомок наследует все свойства и поведение своего родителя, а также может содержать новые свойствами и особенностями поведения. Графически данное отношение обозначается сплошной линией со стрелкой в форме не закрашенного треугольника, которая указывает на родительский вариант использования (рис. 1.3). Эта линия со стрелкой имеет специальное название - стрелка "обобщение".

****

*Рис. 1.3*. Пример графического изображения отношения обобщения между вариантами использования

***Г. Отношение включения*** (*include relationship)* имеет место, когда экземпляр одного варианта использования в процессе выполнения включает в последовательность своего поведения экземпляр второго варианта использования, после чего выполняет последовательность действий, определяющую поведение этого экземпляра варианта использования, по окончании которой продолжает выполнение действий своего поведения.

Графически данное отношение обозначается пунктирной линией со стрелкой, направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом "include" ("включает"), как показано на рис. 1.4.



*Рис. 1.4*. Пример графического изображения отношения включения между вариантами использования

Использование рассмотренных здесь отношений подчиняется следующим правилам.

1. 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.

2. Отношения расширения, обобщения и включения могут существовать только между вариантами использования, которые определены для одной и той же сущности. Причина этого заключается в том, что поведение некоторой сущности обусловлено вариантами использования только этой сущности.

3. Отношения расширения, обобщения и включения, определенные в пределах одной сущности, могут быть использованы в пределах другой сущности, если обе сущности связаны между собой отношением обобщения.

***2. Пример разработки диаграммы вариантов использования***

В качестве примера рассмотрим моделирование программного обеспечения интернет-магазина.

Актерами являются два субъекта: продавец и покупатель. Оба являются пользователями сервиса "Оформить заказ на покупку товара". 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. 1.5).

****

*Рис. 1.5.* Исходная диаграмма вариантов использования для примера разработки системы продажи товаров по каталогу

На следующем этапе разработки данной диаграммы вариант использования "Оформить заказ на покупку товара" может быть уточнен на основе введения в рассмотрение четырех дополнительных вариантов использования. Это следует из более детального анализа процесса продажи товаров, что позволяет выделить в качестве отдельных сервисов такие действия, как обеспечить покупателя информацией о товаре, согласовать условия оплаты товара и заказать товар со склада. Вполне очевидно, что указанные действия раскрывают поведение исходного варианта использования в смысле его конкретизации, и поэтому между ними будет иметь место отношение включения.

С другой стороны, продажа товаров по каталогу предполагает наличие самостоятельного информационного объекта - каталога товаров, который в некотором смысле не зависит от реализации сервиса по обслуживанию покупателей. В нашем случае, каталог товаров может запрашиваться покупателем или продавцом при необходимости выбора товара и уточнения деталей его продажи. Вполне резонно представить сервис "Запросить каталог товаров" в качестве самостоятельного варианта использования.

Полученная в результате последующей детализации уточненная диаграмма вариантов использования будет содержать 5 вариантов использования и 2-х актеров (рис. 1.6), между которыми установлены отношения включения и расширения.

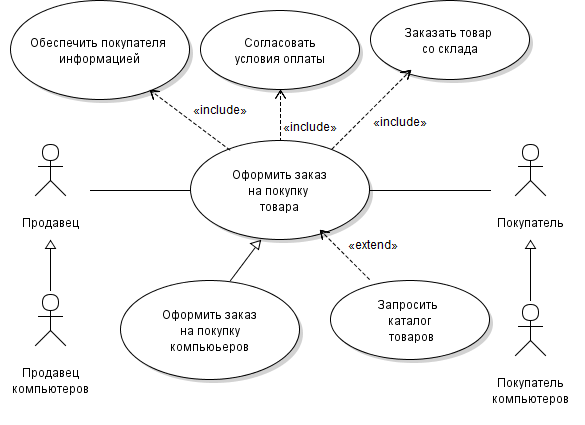
****

*Рис. 1.6.* Уточненный вариант диаграммы вариантов использования для примера системы продажи товаров по каталогу

Полученная таким образом диаграмма вариантов использования, так же может быть далее детализирована путем последующей детализации и структуризации ее отдельных компонентов, например, добавления таких вариантов использования как регистрации пользователей, согласования вариантов доставки, начисление бонусов и скидок и т.д.

Кроме того детализация может быть выполнена на основе установления дополнительных отношений типа отношения "обобщение-специализация" для уже имеющихся компонентов. Так, самостоятельное значение и специфические особенности может иметь отдельная категория товаров - компьютеры. В этом случае диаграмма может быть дополнена вариантом использования "Оформить заказ на покупку компьютера" и актерами "Покупатель компьютера" и "Продавец компьютеров", которые связаны с соответствующими компонентами диаграммы отношением обобщения (рис. 1.7).

Полученная таким образом диаграмма отражает принцип наследования, как фундаментальный принцип объектно-ориентированного программирования. Действительно, на диаграмме отсутствуют изображения линий отношения ассоциации между актером "Продавец компьютеров" и вариантом использования "Оформить заказ на покупку компьютера", а также между актером "Покупатель компьютера" и вариантом использования "Оформить заказ на покупку компьютера". Наличие отношения обобщения между соответствующими компонентами позволяет им наследовать отношение ассоциации от своих предков, а значит, эти линии отношения ассоциации присутствуют на данной диаграмме в скрытом виде.



*Рис. 1.7.* Один из вариантов последующего уточнения диаграммы вариантов использования для примера рассматриваемой системы продажи

***3. Рекомендации по разработке диаграмм вариантов использования***

1. Поскольку модель вариантов использования должна обеспечивать наиболее точное представление системы, процесс ее разработки имеет итерационный характер - допускается ее модификации по мере разработки других диаграмм,

2. На диаграмме должны быть явно определены все сервисы – система, по определению, не может выполнять сервисов, отсутствующих на диаграмме, не зависимо от того, является диаграмма средством разработки или анализа.

3. Не смотря на то, что любой из вариантов использования может быть подвергнут дальнейшей декомпозиции на множество подвариантов, которые образуют исходную сущность, рекомендуемое общее количество актеров в модели - не долее 20, а вариантов использования - не более 50. В противном случае модель теряет свою наглядность и, возможно, заменяет собой одну из некоторых других диаграмм.

**Содержание отчета**

1. Краткое описание выбранной системы (приложение 2).

2. Исходная и уточненная диаграммы вариантов использования, снабженные примечаниями и количественной оценкой (приложение 1).

**Контрольные вопросы**

1. Какие элементы диаграммы представляют функциональность проектируемой системы?

2. Какие элементы диаграммы представляют внешние сущности?

3. Для какого типа отношения не указывается направление линии, обозначающей это отношение?

4 .Когда недопустимо наличие связи между двумя вариантами использования?

5. Какие варианты отношений могут существовать между актерами?

**ЛАБОРАТОРНАЯ РАБОТА № 2**

**ДИАГРАММА КЛАССОВ**

**Цель работы**

Сформировать навыки разработки статической модели программного обеспечения на основе объектно-ориентированного подхода.

**Задание**

Разработать диаграмму классов (class diagram) программного обеспечения для разработанной ранее диаграммы вариантов использования (в лабораторной работе № 1) и написать комментарии к ней.

**Порядок выполнения**

***1. Описание диаграммы***

Диаграмма классов занимает центральное место в логической модели ПО и используется для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Эта диаграмма может рассматриваться и стоиться, как дальнейшее развитие концептуальной модели проектируемой системы (лабораторная работа № 1).

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы представляющие классы, дуги задают различные типы структурных отношений, которыми они связаны.

***1.1. Классы***

Класс (class) обозначает множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 2.1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (рис. 2.1, а). По мере проработки отдельных компонентов диаграммы, описания классов дополняются атрибутами и операциями (рис. 2.1, б).

Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций.

****

Рис. 2.1. Графическое изображение класса на диаграмме классов

Примеры графического изображения классов на диаграмме классов приведены на рис. 2.2. В первом случае для класса "Прямоугольник" (рис. 2.2, а) указаны только его атрибуты - точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис. 2.2, б) указаны только его операции, секция атрибутов оставлена пустой.

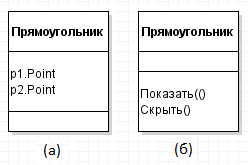


Рис.2.2. Примеры графического изображения классов на диаграмме

***1.1.1. Имя класса.***

Имя класса указывается в первой верхней секции прямоугольника, записывается по центру секции полужирным шрифтом и начинается с заглавной буквы. В качестве имен классов рекомендуется использовать существительные, записанные (по практическим соображениям) без пробелов. Имя класса должно быть уникальным в пределах пакета.

В этой же секции могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и, соответственно, от которых он наследует свойства и методы. Кроме того здесь же может приводиться информация о разработчике данного класса, статус состояния разработки, другие общие свойства этого класса, имеющие отношение к другим классам диаграммы. Перед именем класса через разделитель - двойное двоеточие (::), может быть записано имя пакета, к которому относится данный класс:

*<Имя\_пакета>::<Имя\_класса>.*

Например, если определен пакет с именем "Банк", то класс "Счет" в этом банке может быть записан в виде: "Банк::Счет".

***1.1.2 Атрибуты класса***

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. Каждому атрибуту класса соответствует строка текста вида:

<*квантор видимости*><*имя атрибута*>[*кратность*]:

<*тип атрибута*> = <*исходное значение*>{*строка-свойство*}

*Имя атрибута* является обязательным и записывается со строчной (малой) буквы. Оно используется в качестве идентификатора, и поэтому должно быть уникальным в пределах данного класса.

*Квантор видимости* может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов:

1. Символ "+" обозначает атрибут с областью видимости типа общедоступный (*public*). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.

2. Символ "#" обозначает атрибут с областью видимости типа защищенный (*protected*). Атрибут с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса.

3. Символ "-" обозначает атрибут с областью видимости типа закрытый (*private*). Атрибут с этой областью видимости недоступен или невиден для всех классов без исключения.

Вместо условных графических обозначений можно записывать соответствующее ключевое слово: *public, protected, private*. Если квантор видимости опущен, то это означает, что видимость атрибута не указывается и может быть любой[[1]](#footnote-1).

*Кратность атрибута* характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В общем случае кратность записывается в форме строки текста в квадратных скобках после имени соответствующего атрибута:

[*нижняя\_граница\_1 .. верхняя\_граница\_1, нижняя\_граница\_2..* *верхняя\_грашца\_2, ..., нuжняя\_гpaнuцa\_k … верхняя\_граница\_k*],

Нижние и верхние границы интервалов включаются в значение кратности. В качестве верхней границы может использоваться специальный символ "\*", который означает произвольное положительное целое число и означает неограниченное сверху значение кратности соответствующего атрибута. Если в качестве кратности указывается единственное число, то кратность атрибута принимается равной данному числу. Если же указывается единственный знак "\*", то это означает, что кратность атрибута может быть произвольным положительным целым числом или нулем.

Например,

1) [0..1] означает, что кратность атрибута может принимать значение О или 1. При этом 0 означает отсутствие значения для данного атрибута;

2) [0..\*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 0. Эта кратность может быть записана короче в виде простого символа - [\*];

3) [1.:\*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1;

4) [1..5] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5;

5) [1..3,5,7] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 5, 7;

6) [1..3,7.. 10] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 7, 8, 9, 10;

7) [1..3,7..\*] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, а также любое положительное целое значение большее или равное 7.

Если кратность атрибута не указана, то по умолчанию принимается ее значение равное 1..1, т. е. в точности 1.

*Тип атрибута* записывается с прописной (большой) буквы и представляет собой выражение, семантика которого определяется языком программирования, который предполагается использовать для реализации данной модели. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс.

Имена и типы атрибутов классов могут быть заданы, например, следующим образом.

1. Цвет: Соlоr - здесь цвет является именем атрибута, Color - именем типа данного атрибута.

2. Имя\_сотрудника [1..2] : String. Кратность указывает на то, что имя сотрудника может состоять из двух слов (например, имя и отчество)

3. Видимость:Boolean - здесь указано, что возможными значениями атрибута "Видимость" является одно из двух логических значений: истина (true) или ложь (false), например, значение истина может соответствовать наличию графического изображения на экране монитора, а значение ложь - его отсутствию.

4. Форма : Многоугольник - здесь имя атрибута "Форма" может характеризовать тип геометрической фигуры на плоскости, например, треугольник, прямоугольник, ромб, пятиугольник и любой другой многоугольник, но не окружности или эллипса.

*Исходное значение* служит для задания некоторого начального значения для соответствующего атрибута в момент создания отдельного экземпляра класса, например, дополнительно исходные значения атрибутов могут быть указаны следующим образом.

1. Цвет:Соlоr = (255, 0, 0) - в RGB-модели цвета это соответствует чистому красному цвету в качестве исходного значения для данного атрибута.

2. Видимость:Вооlеаn = true может соответствовать ситуации, когда в момент создания экземпляра класса создается видимое на экране монитора окно, соответствующее данному объекту.

3. Форма:Многоугольник = Прямоугольник – создается объект типа "Прямоугольник".

Если исходное значение не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса

*Строка-свойство* служит для указания значений атрибутов, которые они должны принимать при создании экземпляров данного класса и которые не могут быть изменены в программе при работе с этими объектами. Например, строка-свойство в записи атрибута

*заработная\_плата:Currency = {$500}*

обозначает фиксированное значение заработной платы для каждого объекта класса "Сотрудник".

Отсутствие строки-свойства означает, что значение соответствующего атрибута может быть изменено в программе, например, запись данного атрибута в виде

*заработная\_плата: Currency = $500*

означает, что при создании нового экземпляра "Сотрудник" (при приеме на работу нового сотрудника) заработная плата по умолчанию устанавливается в $500, которая в дальнейшем в программе может быть изменена.

***1.1.3. Операции класса***

В третьей сверху секции прямоугольника записываются операции или методы класса. Операция (*operation*) представляет собой некоторый сервис, предоставляемый каждым экземпляром класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса. Каждой операции класса соответствует строка, текста вида:

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

<*выражение типа возвращаемого значения*>{*строка-свойство*}

*Имя операции* является обязательным и записывается со строчной буквы (малой) буквы, за которым обязательно должны следовать круглые скобки (возможно без заключенного в них текста). Имя операции используется в качестве идентификатора, и поэтому должно быть уникальным в пределах данного класса.

*Квантор видимости*, как и в случае атрибутов класса, может принимать одно из трех возможных значений и, соответственно, отображается при помощи специальных символов.

1. Символ "+" обозначает операцию с областью видимости типа общедоступный (public).

2. Символ "#" обозначает операцию с областью видимости типа защищенный (protected).

3. Символ "-" используется для обозначения операции с областью видимости типа закрытый (private).

Вместо условных графических обозначений также можно записывать соответствующее ключевое слово: public, protected, private[[2]](#footnote-2). Квантор видимости для операции может быть опущен и видимость операции может быть любой.

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

<*направление* *параметра*><*имя параметра*>:<*выражение типа*>=<*значение*

*параметра по умолчанию*>,

где *направление* *параметра* одно из трех ключевых слов: in - параметр является входным; out - выходным; inout – и тем и другим (по умолчанию, если вид параметра не указан, - in); *имя параметра* - идентификатор соответствующего формального параметра; *выражение типа* - зависимая от конкретного языка программирования спецификация типа возвращаемого значения для соответствующего формального параметра; *значение параметра по умолчанию* - выражение для значения формального параметра, синтаксис которого зависит от конкретного языка программирования и подчиняется принятым в нем ограничениям.

*Выражение типа возвращаемого значения* также является зависимой от языка реализации спецификацией типа или типов значений параметров, которые возвращаются объектом после выполнения соответствующей операции. Двоеточие и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения. Для указания кратности возвращаемого значения данная спецификация может быть записана в виде списка отдельных выражений.

Список формальных параметров и тип возвращаемого значения могут не указываться.

*Строка-свойство* служит для указания значений свойств, которые могут быть применены к данному элементу.

**{*query*}** ({*запрос*}) – такая операция не может изменять состояние системы и, соответственно, не имеет никакого побочного эффекта, в противном случае операция может изменять состояние системы.

**{*concurrency*}** указывает на возможность параллельного выполнения операции:

{*concurrency = sequential*} - последовательная операция, для которой должно быть обеспечено единственное выполнение - одновременное выполнение вместе с ней других операций недопустимо;

{*concurrency = concurrent*} - параллельная операция, для которой допустимо одновременное выполнение с другими операциями;

{*concurrency = guarded*} – охраняемая операция, обращения к которой должны быть строго упорядочены во времени, и приняты дополнительные меры по контролю исключительных ситуаций на этапе ее выполнения.

Если значение параллельности опущено (строка-свойство имеет вид {*concurrency*}) считается, что операция требует последовательного выполнения.

**{*abstract*}** – абстрактная операция - объявленная, но не реализованная в данном классе, которая будет использована в соответствующем классе-потомке.

Кроме того в форме строки-свойства (в фигурных скобках) могут быть дополнительно указаны особенности реализации операции, представляющие собой ее формальную спецификацию на языке реализации модели.

Строка-свойство не является обязательной, она может отсутствовать, если никакие свойства не специфицированы.

Примеры записи операций.

1. *+создать()* - может обозначать абстрактную операцию по созданию отдельного объекта класса, которая является общедоступной и не содержит формальных параметров, операция не возвращает никакого значения после своего выполнения.

2. *+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.

3. *запросить\_счет\_клиента(номер\_счета:Long1пt):Сиггепсу* - 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"). Результатом выполнения этой операции является некоторое число, записанное в принятом денежном формате (например, $1,500.00).

4. *выдать\_сообщение():{"Ошибка деления на ноль"} –* в результате выполнения этой операции на экране монитора может появиться сообщение, указанное в строке-свойстве.

***1.2. Отношения между классами***

Кроме внутреннего устройства или структуры классов на диаграмме указываются различные отношения между классами. При этом совокупность типов таких отношений фиксирована в языке UML и предопределена семантикой этих типов отношений. Базовыми отношениями между классами или связями в языке UML являются:

1) отношение зависимости (*dependency relationship*);

2) отношение ассоциации (*association relationship*;

3) отношение обобщения (*inherits\_from relationship*);

4) отношение реализации (*realization relationship*)

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов. Специальной формой или частным случаем отношения ассоциации является отношение агрегации, которое, в свою очередь, тоже имеет специальную форму - отношение композиции.

**А. Отношение зависимости** (*dependency relationship*). Отношение зависимости является наиболее общей формой отношения. Оно имеет место всегда, когда изменение спецификации одного класса (класса-источника) может повлиять на работу другого класса (класса-клиента), но не наоборот.

Отношение зависимости графически изображается пунктирной линией со стрелкой, направленной от класса-клиента зависимости к независимому классу или классу-источнику. На Рис. 2.3 изображены два класса: Класс\_А и Класс\_Б, при этом Класс\_Б является источником некоторой зависимости, а Класс\_А - клиентом этой зависимости.

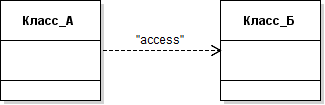
****

Рис. 2.3. Графическое изображение отношения зависимости на диаграмме классов

Рядом со стрелкой может быть указано имя отношения и в кавычках один из следующих стереотипов:

1) "access" - служит для обозначения доступности открытых атрибутов и операций класса-источника для классов-клиентов;

2) "bind" - класс-клиент может использовать некоторый шаблон для своей последующей параметризации;

3) "derive" - атрибуты класса-клиента могут быть вычислены по атрибутам класса-источника;

4) "import" - открытые атрибуты и операции класса-источника становятся частью класса-клиента, как если бы они были объявлены непосредственно в нем;

5) "refine" - указывает, что класс-клиент служит уточнением класса-источника в силу причин исторического характера, когда появляется дополнительная информация в ходе работы над проектом.

Стереотип и имя отношения указывать не обязательно.

**Б. Отношение ассоциации** *(association relationship).* Отношение ассоциации имеет место, когда объекты одного класса связаны с объектами другого класса таким образом, что можно перемещаться от объектов одного класса к другому. Отношение ассоциации обозначается сплошной линией, стрелка указывает на порядок следования классов. Для отношения может быть указано имя ассоциации, и кратность классов. Имя записывается с заглавной (большой) буквы; кратность классов обозначается аналогично кратности атрибутов (рис. 2,4)

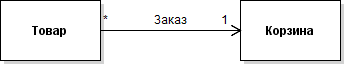


Рис. 2.4. Диаграмма классов, находящихся в отношении бинарной ассоциации

Имя ассоциации и кратность классов указывать не обязательно.

**В. Отношение агрегации** (*aggregation* *relationship)* являетсячастным случаем отношения ассоциации. Отношение агрегации 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. 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.[[3]](#footnote-3)

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой не закрашенный внутри ромб, который указывает на класс, представляющий собой "целое". Остальные классы являются его "частями" (рис. 2.5).

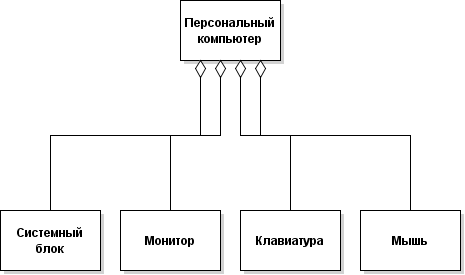


Рис. 2.5. Диаграмма классов, находящихся в отношения агрегации

**Г. Отношение композиции** (*Composition* *relationship)* является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого, и эти части существуют до тех пор, пока существует целое и прекращают свое существование вместе с целым.

Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб, который указывает на класс, представляющий собой "целое". Остальные классы являются его "частями" (рис. 2.6).

В качестве дополнительных обозначений для отношений агрегации и композиции могут использоваться дополнительные обозначения, применяемые для отношения ассоциации. А именно, указание кратности класса ассоциации и имени данной ассоциации, которые не являются обязательными.

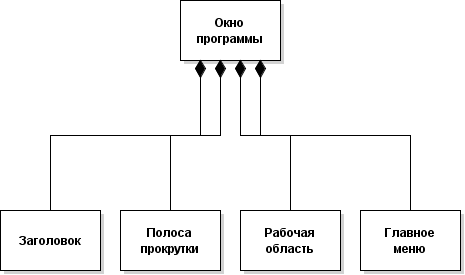


Рис. 2.6. Диаграмма классов, находящихся в отношения композиции

**Д. Отношение обобщения**. (*inherits\_from relationship)* является обычным таксономическим отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). 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. Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие - на более специальный класс (класс-потомок или подкласс) (рис. 2.7).

Рядом со стрелкой может быть указано имя отношения и в фигурных скобках дополнительные ограничения отношения:

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

2) {disjoint} - означает, что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов;

3) {incomplete} - означает, что на диаграмме указаны не все классы-потомки;

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

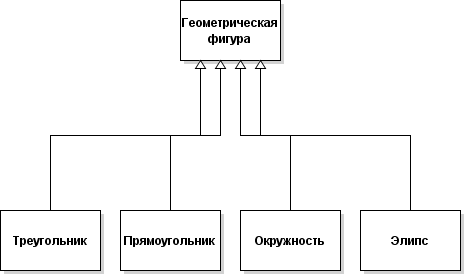


Рис. 2.7. Диаграмма классов, находящихся в отношения обобщения

***1.3. Интерфейсы***

Интерфейс (interface) служит для спецификации параметров модели, которые видимы извне без указания их внутренней структуры[[4]](#footnote-4). Формально интерфейс эквивалентен абстрактному классу без атрибутов и методов с наличием только абстрактных операций.

Для изображения интерфейсов используется специальный графический символ - прямоугольник класса с ключевым словом или стереотипом "interface". При этом секция атрибутов у прямоугольника отсутствует, а указывается только секция операций. Имя (если оно записывается на английском), должно начинаться с заглавной буквы "I", например, ISecurelnformation, ISensor. Отношение изображается пунктирной линией с не закрашенным треугольником, направленным на данный интерфейс (рис. 2.8).

***1.4. Пакеты***

Пакет (*package*) позволяет изолировать имена содержащихся в нем классов – два класса могут иметь одинаковые имена, если они принадлежат разным пакетам.

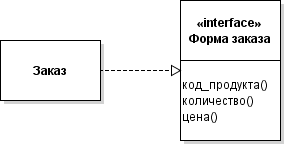
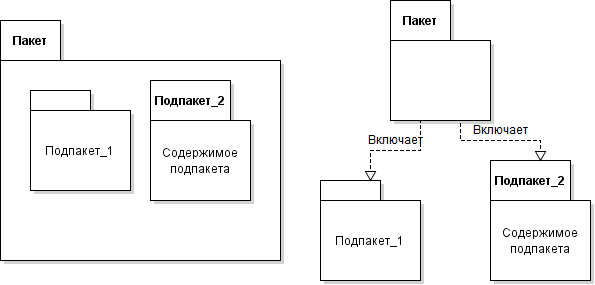


Рис. 2.8. Пример графического изображения интерфейса на диаграмме классов

Пакет на диаграммах представляется в виде большого прямоугольника с небольшим прямоугольником, присоединенным к левой части верхней стороны первого. Имя пакета может записываться в верхнем маленьком прямоугольнике, или в большом, если он не содержит ничего иного. Имя пакета должно быть уникальным в пределах рассматриваемой модели. Перед именем пакета допускается (но не является обязательным) использование следующих стереотипов: facade, framework, stub и topLevel.

Каждый пакет владеет всеми элементами, которые включены в него. Вложенные в пакет пакеты называются подпакетами, и все элементы подпакета считаются вложенными в более общий пакет (рис. 2.18а). Тем самым для пакетов задается отношение вложенности или включения пакетов, которое представляет собой иерархию и может быть представлено в виде отношения обобщения (рис. 2.18б).



а б

Рис. 2.18. Графическое изображение вложенности пакетов друг в друга

***2. Рекомендации по построению диаграмм классов***

На начальных этапах разработки диаграмм целесообразно составить словесное описание решаемой задачи или всей предметной области. В таком описании, в большинстве случаев, классам соответствуют существительные; отношениям – глаголы; значениям атрибутов – перечисляемые или задаваемые значения некоторых свойств. Например, из предложения «система бронирования для продажи билетов на представления в различных театрах» можно выделить классы «Бронирование», «Система», «Билет», «Представление» и «Театр». Глаголы могут характеризовать размещение («находятся в …», «содержит …», …); передачу информации («разговаривает с …», «сообщает о …», …); направленные действия («управляет …»; «сотрудничает с …», «работает на …», ...); … В отличие от классов и отношений непосредственно выделить атрибуты классов , как правило, не удается. Атрибутом может являться цвет (если некоторое свойство может иметь, например, значение синий, зеленый и т.д,), возраст (или дата рождения), пол (мужской/женский), состояние (включен/выключен) и т.д.

Полученная описанным здесь способом диаграмма, естественно, будет нуждаться в уточнении: добавлении и (или) удалении классов, их атрибутов и операций, а также отношений. Если два класса выражают одно и то же понятие, (например, покупатель и пользователь интернет-магазина), один из таких классов следует удалить, оставив тот, который лучше описывает сущность понятия. Классы, характеризующие индивидуальные объекты следует сделать атрибутами (например, имя, адрес, вес…). И наоборот, если некоторое свойство должно существовать независимо от носителя, его следует сделать классом (например, пункт выдачи товаров интернет-магазина). Если какой-либо атрибут класса отличается от всех остальных и с ними не связан, то содержащий его класс следует разделить на 2 класса. Операция, обладающая собственными чертами, должна быть представлена классом (например, выдача заказа в интернет-магазине должна быть классом с атрибутами дата, номер, сумма и способ оплаты и т.д.). Диаграммы верхних уровней иерархии не должны содержать конструкций, относящихся к реализации (например, процесс, алгоритм, прерывание и т.д.). Следует исключить атрибуты, описывающие внутреннее состояние объекта, невидимое снаружи. Если удален класс, связанный отношением, то удалить следует и отношение.

После описанных здесь действий весьма целесообразно распределить классы и отношения по пакеты. Для этого ищутся классы, являющиеся единственным соединением двух частей модели и не имеющие между собой других связей (это так называемые cut points – точки сочленения). Они то и образуют «мост» между двумя пакетами. Внутри пакеты должны содержать сведенное к минимуму число пересечений.

Процесс разработки диаграммы классов имеет итерационный характер, и после разработки диаграмм состояний (лабораторная работа № 3) и диаграмм последовательности (лабораторная работа № 5) могут потребоваться дополнительные изменения.

**Содержание отчета**

1. Диаграмма классов для разработанной ранее диаграммы вариантов использования (Лабораторная работа № 1) и ее количественная оценка.

2. Описание атрибутов и операций классов разработанной диаграммы с указанием реализуемых ими прецедентов диаграммы Use Case.

**Контрольные вопросы**

1. Какие свойства атрибутов класса могут быть представлены на диаграмме?

2. Какие свойства операций класса могут быть представлены на диаграмме?

3. Какие свойства параметров операций класса могут быть представлены на диаграмме?

4. Какие свойства отношения зависимости могут быть представлены на диаграмме?

5. Какие свойства отношения обобщения могут быть представлены на диаграмме?

**ЛАБОРАТОРНАЯ РАБОТА № 3**

**ДИАГРАММА СОСТОЯНИЙ**

**Цель работы**

Сформировать навыки разработки модели поведения проектируемой информационной системы.

**Задание**

Разработать диаграмму состояний (statechart (state) diagram) системы, обеспечивающей функциональность, заданную диаграммой использования (лабораторная работа № 1), и имеющей структуру, заданную диаграммой классов (лабораторная работа № 2) и написать комментарии к ней.

**Порядок выполнения**

***1. Описание диаграммы***

Каждая прикладная система характеризуется не только структурой составляющих ее элементов, которая представляется, например, диаграммой классов (см. лабораторная работа № 2), но и некоторым поведением или функциональностью. Для общего представления функциональности моделируемой системы предназначены диаграммы вариантов использования (см. лабораторная работа № 1), которые на концептуальном уровне описывают поведение системы в целом. Для более детального представления поведения системы, которое обеспечивает необходимую функциональность, используются диаграммы состояний.

Диаграммы состояний позволяют описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение системы в целом или ее элемента, как некоторого объекта, и представляют динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий[[5]](#footnote-5).

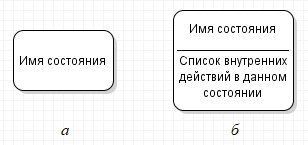
Основными элементами диаграммы состояний являются состояние и переход.

***1.1. Состояние***

В языке UML состояние понимается как отдельная ситуация, 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. Однако не каждый атрибут класса может характеризовать его состояние. Как правило, имеют значение только такие свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения. В этом случае состояние будет характеризоваться некоторым инвариантным условием, включающим в себя только значимые для поведения класса атрибуты и их значения.

Например, инвариант может представлять статическую ситуацию, когда объект находится в состоянии ожидания возникновения некоторого внешнего события. С другой стороны, инвариант используется для моделирования динамических аспектов, когда в ходе процесса выполняются некоторые действия. В этом случае моделируемый элемент переходит в рассматриваемое состояние в момент начала соответствующей деятельности и покидает данное состояние в момент ее завершения.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рис. 3.1).



*Рис.3.1.* Изображение состояний на диаграмме состояний

***Имя состояния*** представляет собой строку текста, которая раскрывает содержательный смысл данного состояния. Имя всегда записывается с заглавной буквы. Поскольку состояние системы является составной частью процесса ее функционирования, рекомендуется в качестве имени использовать глаголы в настоящем времени (звенит, печатает, ожидает) или соответствующие причастия (занят, свободен, передано, получено).

***Список внутренних действий.*** Эта секция 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, которую определяет *выражение\_действия*. При этом *выражение\_действия* может использовать любые атрибуты и связи, которые принадлежат области имен или контексту моделируемого объекта. Если *выражение\_действия* отсутствует, то разделитель в виде наклонной черты '/' может не указываться.

*Метка\_действия* может иметь одно из следующих т.н. фиксированных значений[[6]](#footnote-6):

1) *entry* - эта метка указывает на действие, которое выполняется в момент входа в данное состояние (входное действие);

2) *exit* - эта метка указывает на действие, которое выполняется в момент выхода из данного состояния (выходное действие);

3) *do* - эта метка определяет выполняющуюся деятельность ("do activity") в течение всего времени, пока объект находится в данном состоянии;

Во всех остальных случаях *метка\_действия* идентифицирует событие, которое запускает соответствующее *выражение\_действия*. Эти события называются внутренними переходами и семантически эквивалентны переходам в само это состояние, за исключением той особенности, что выход из этого состояния или повторный вход в него не происходит, таким образрм, действия входа и выхода не выполняются.

Пример состояния с непустой секцией внутренних действий приведен на рис. 3.2.

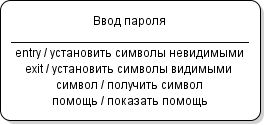


Рис. 3.2. Состояние «Ввод пароля»

Диаграмма состояний должна содержать два специальных состояния, которые представлены на рис. 3.3.

Рис. 3-3_1.png

*Рис. 3.3.* Графическое изображение начального и конечного состояний на диаграмме состояний: *а* – начальное состояние; *б* – конечное состояние

***Начальное состояние.*** В этом состоянии объект находится по умолчанию в начальный момент времени. Оно служит для обозначения состояния, с которого начинается процесс изменения состояний.

***Конечное состояние.*** В этом состоянии объект будет находиться по умолчанию в конечный момент времени. Оно служит для обозначения состояний, в которых завершается процесс изменения состояний.

***1.2. Переход***

Переход отображает смену одного состояния другим. На диаграмме состояний переход изображается сплошной линией со стрелкой, направленной из состояния, которое называется исходным в состояние, которое называется целевым. Факт смены одного состояния другим принято называть срабатыванием перехода. Переход срабатывает в результате наступления некоторого события. Если этим событием является завершение всех операций исходного состояния, то переход называется нетриггерным, в отличие от триггерного перехода, который срабатывает в результате возникновения некоторого внешнего (по отношению к рассматриваемой диаграмме состояний) события.

Переход может быть помечен строкой текста следующего формата:

*<имя события>’(‘< список параметров, разделенных запятыми>’)’[‘< сторожевое условие>’]’< список выражений действий, разделенных точкой с запятой (;)>*

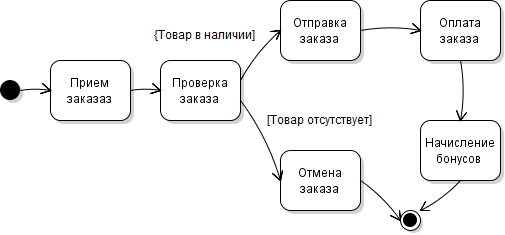
Обязательным параметром является только имя события.

В качестве ***события*** ***(event)*** могут выступать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. После имени события могут следовать круглые скобки для явного задания параметров соответствующего события-триггера (например, пользователь инициирующий действие).

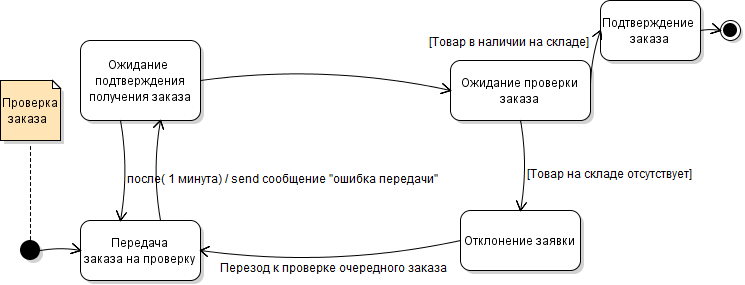
Если при срабатывании перехода возможно ветвление, в имени перехода используется ***сторожевое условие (guard condition)***, которое записывается в прямых скобках после события-триггера. Сторожевое условие должно определять семантику срабатывания соответствующего перехода. Переход должен будет сработать, если сторожевое условие принимает значение «истина», и переход не сможет сработать в противном случае. При этом никакие два сторожевых условия не должны одновременно принимать значение "истина". Вычисление истинности сторожевого условия происходит после возникновения события-триггера, инициирующего соответствующий переход.

После сторожевого может следовать ***список выражений действий, (action expression)*** которые должны будут выполниться при срабатывании перехода, и до начала каких бы то ни было действий в целевом состоянии. При создании диаграмм состояний на нижних уровнях иерархии модели системы (при описании ее отдельных компонент) выражения действия записываются на одном из языков программирования, который предполагается использовать для реализации модели. Предполагается, что действия не будет прервано до тех пор, пока оно не закончится (действие представляет собой атомарную операцию). Однако оно может оказывать влияние, как на сам объект, так и на его окружение, если это с очевидностью следует из контекста модели[[7]](#footnote-7).

Пример диаграмм состояний приведен на рис. 3.4 и рис. 3.5.



*Рис. 3.4.* Диаграмма состояний интернет магазина



*Рис. 3.5.* Диаграммы проверки заказа в интернет магазине

***2. Рекомендации по построению диаграмм состояний***

В качестве начального варианта диаграммы состояний, если нет очевидных соображений по поводу состояний объекта, можно воспользоваться суперсостояниями простой дихотомии: «исправен - неисправен», «активен - неактивен», «ожидание - реакция на внешние действия» , рассматривая их как составные и уточняя их (детализируя их внутреннюю структуру) по мере рассмотрения логики поведения системы.

При выделении состояний и переходов следует помнить, что длительность срабатывания отдельных переходов должна быть существенно меньшей, чем нахождение моделируемого объекта в соответствующих состояниях. Каждое из состояний должно характеризоваться определенной устойчивостью во времени, и из каждого состояния на диаграмме не может быть самопроизвольного перехода в какое бы то ни было другое состояние. Все переходы должны быть явно специфицированы, в противном случае построенная диаграмма состояний является либо неполной, либо ошибочной.

При разработке диаграммы состояний нужно постоянно следить, чтобы объект в каждый момент мог находиться только в единственном состоянии. Если это не так, то данное обстоятельство может быть как следствием ошибки, так и неявным признаком наличия параллельности у поведения моделируемого объекта.

Следует обязательно проверять, что никакие два перехода из одного состояния не могут сработать одновременно (требование отсутствия конфликтов у переходов). Наличие такого конфликта может служить признаком ошибки,[[8]](#footnote-8) и необходимо либо ввести дополнительные сторожевые условия, либо изменить существующие, чтобы исключить конфликт переходов.

**Содержание отчета**

1. Диаграммы состояний для разработанных ранее диаграмм вариантов использования (лабораторная работа № 1) и диаграммы классов (лабораторная работа № 1) и их количественные оценки.

2. Описание состояний представленных на разработанной диаграмме с указанием выполняемых в каждом состоянии методов..

**Контрольные вопросы**

1. Результатом детализации какой диаграммы является диаграмма состояний?

2. Каким образом можно обозначить первое и последнее выполняемые действия в заданном состоянии?

3. Что может инициализировать переход из одного состояния в другое?

4. Когда наличие сторожевого условия является обязательным в описании перехода?

5. Когда происходит вычисление сторожевых условий?

6. Какие действия являются не допустимыми в списке действий в описания перехода?

**ЛАБОРАТОРНАЯ РАБОТА № 4**

**ДИАГРАММА ДЕЯТЕЛЬНОСТИ**

**Цель работы**

Сформировать навыки разработки логики и последовательности действий в информационной системе в соответствии с функциональными требованиями.

**Задание**

Разработать диаграмму деятельности (activity diagram) системы, соответствующую диаграмме вариантов использования (лабораторная работа № 1) и написать комментарии к ней.

**Порядок выполнения**

***1. Описание диаграммы***

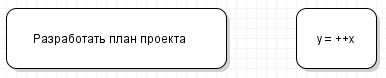
При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний (лабораторная работа № 3), но и детализировать особенности алгоритмической и логической реализации операций, выполняемых системой. В используемых для этой цели традиционных блок-схемах алгоритмов (см. ГОСТ 19.701 – 90 (ИСО 5807 – 85) «Схемы алгоритмов …») отсутствуют средства для представления параллельных процессов и их синхронизации.

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 и в отсутствии на переходах описаний 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, вершинами которого являются состояния действия, а дугами - переходы от одного состояния действия к другому. При этом каждое состояние может являться выполнением операции некоторого класса или ее части, что позволяет использовать диаграммы деятельности для описания реакций на внутренние события системы.

***1.1. Состояние действия***

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. Внутри этой фигуры записывается выражение действия (action expression), которое должно быть уникальным в пределах одной диаграммы деятельности.



*а б*

*Рис. 4.1.* Графическое изображение состояния действия: *а* простое действие; *б* выражение

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. 4.1а). 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).

Диаграмма действия может описывать поведение, на которое оказывают влияние внешние события, происходящие за пределами данной системы. Для этого используются символ *передача сигнала (send signal action)* и символ п*рием события (receive event action)*. Первый символ (рис. 4.2а) обозначает действие, которое на основе своих входов создает экземпляр сигнала и передает его внешней Системе, второй (рис. 4.2б) – действие, которое ожидает некоторого события, принимает и обрабатывает полученное сообщение.

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний (см. рис. 4.3). 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.

Рис. 4-2_I.png

*а б*

*Рис. 4.2.* Специальные символы действий

***1.2. Переходы***

Переход, как элемент языка UML, был рассмотрен в лабораторной работе № 3. При построении диаграммы деятельности используются только нетриггерные переходы, т. е. такие, 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 (см. Лабораторную работу № 3). 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.

Графически ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста. 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

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. 4.3). 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.

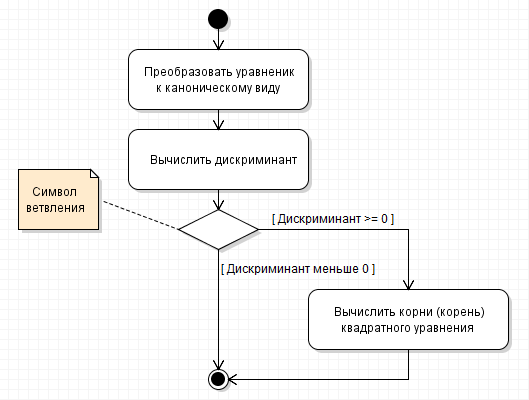
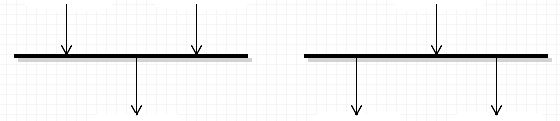


Рис. 4.3. Диаграмма деятельности для алгоритма нахождения корней квадратного уравнения

Несколько входящих переходов и один выходящий переход имеет символ объединения, который так же изображается ромбом.

Один из наиболее значимых недостатков обычных структурных схем алгоритмов (блок-схем) связан с проблемой изображения параллельных ветвей отдельных вычислений. В UML для представления параллельных процессов используется специальный символ для обозначения разделения и слияния параллельных вычислений или потоков управления. Таким символом является прямая черточка.

Как правило, такая черточка изображается отрезком горизонтальной линии, толщина которой несколько шире основных сплошных линий диаграммы деятельности. При этом слияние (concurrent join), имеет несколько входящих переходов и один выходящий (рис. 4.4а) разделение (concurrent fork), наоборот, имеет один входящий переход и несколько выходящих (рис. 4.4б).

****

*а б*

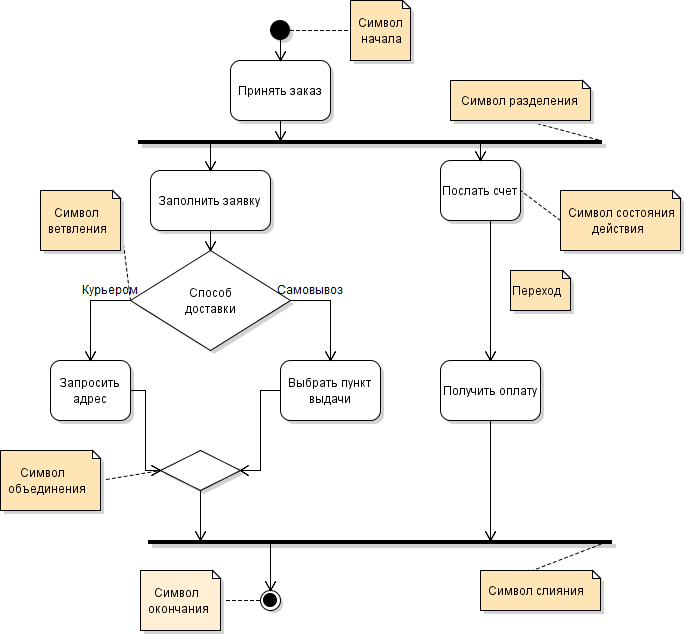
Рис. 4.4. Графическое изображение слияния и разделения параллельных потоков управления: а – слияние; б – разделение

На рис. 4.5 приведен пример диаграммы деятельности с использованием символов разделения и слияния. Символ разделения обозначает, что операции «Заполнить заявку» и «Отправить отчет» выполняются параллельно и порядок их выполнения не имеет значения – сначала можно вручить товар и потом получить его оплату, либо оформлять заявку только при наличии предоплаты. Символу разделения должен соответствовать символ слияния. В рассматриваемом случае заказ не закрывается до тех пор пока он не доставлен и не оплачен.

***2. Рекомендации по построению диаграмм деятельности***

Диаграммы деятельности, как и диаграммы состояний, могут строиться для отдельного класса, варианта использования, отдельной операции класса или целой подсистемы, и процесс их разработки представляет собой последовательность итераций нисходящего и восходящего проектирования.

На начальных этапах проектирования, когда детали реализации деятельностей в проектируемой системе неизвестны, построение диаграммы деятельности начинают с выделения «поддеятельностей», которые в совокупности образуют деятельность подсистем. В последующем, по мере разработки диаграмм классов и состояний, эти «поддеятельности» уточняются в виде отдельных вложенных диаграмм деятельности компонентов подсистем, какими выступают классы и объекты.

****

*Рис.4.5*. Процесс совершения покупки через интернет магазин

Если большинство деталей реализации действий известны заранее, то на основе предшествующего опыта разработки систем-прототипов, строятся диаграммы деятельности для готовых решений. В последующем такие диаграммы вкладывается в более общие диаграммы деятельности для подсистемы и системы в целом, сохраняя свой уровень детализации.

Таким образом, процесс анализа и проектирования информационных систем с использованием диаграмм деятельности представляется как последовательность итераций нисходящей и восходящей разработки отдельных диаграмм. Доминирование того или иного из направлений разработки определяется особенностями конкретного проекта и его новизной.

Признаком завершения отдельных этапов работы над проектом является достижение такого уровня детализации всех диаграмм, которого достаточно для понимания особенностей реализации всех действий и деятельностей[[9]](#footnote-9).

**Содержание отчета**

Диаграммы деятельности для разработанных ранее диаграмм вариантов использования (Лабораторная работа № 1), диаграмм классов (Лабораторная работа № 2) и диаграмм состояний (Лабораторная работа № 3), с указанием методов классов, образующих действие, комментарии и количественная оценка.

**Контрольные вопросы**

1. Какой тип перехода используется в диаграммах деятельности?

2. Какие символы обозначают взаимодействие проектируемой системы с внешней средой?

3. Что инициализирует переход из одного состояния действия к другому?

4. Каким свойством должны обладать сторожевые условия переходов при выходе из состояния ветвления?

5. Когда срабатывает переход, выходящий из символа слияния?

**ЛАБОРАТОРНАЯ РАБОТА № 5**

**ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ**

**Цель работы**

Сформировать навыки разработки взаимодействия экземпляров классов (объектов) в информационной системе.

**Задание**

Разработать диаграмму последовательности (sequence diagram) системы, соответствующую диаграммам вариантов использования (лабораторная работа № 1) и диаграмме классов (лабораторная работа № 2) и написать комментарии к ней.

**Порядок выполнения**

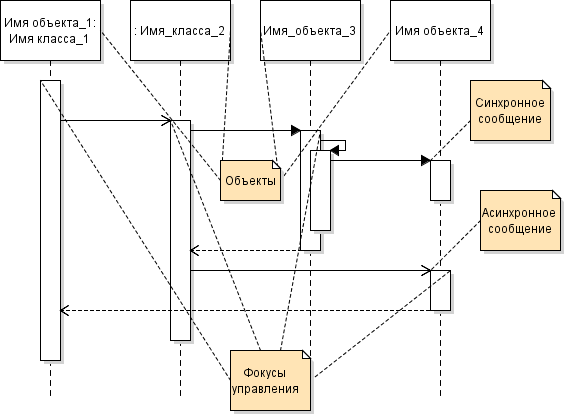
***1. Описание диаграммы***

Кроме последовательности изменения состояний (лабораторная работа № 3) и алгоритма функционирования (лабораторная работа № 4) система определяется еще и взаимодействием образующих ее элементов. В UML такими элементами являются экземпляры классов – объекты. В процессе взаимодействия объекты обмениваются некоторой информацией в форме сообщений. Для описания этого процесса используются диаграммы последовательности.

***1.1. Объекты***

На диаграмме последовательности изображаются только те экземпляры классов (объекты), которые непосредственно участвуют во взаимодействии, и не показываются объекты, с которыми возможны иные статические ассоциации. Диаграмма последовательности имеет как бы два измерения.

Одно - слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни (рис. 1)[[10]](#footnote-10). Внутри прямоугольника записываются имя объекта и имя класса, разделенные двоеточием.[[11]](#footnote-11) Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия (объект 1 на рис. 5.1). Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый степенью активности этих объектов при взаимодействии друг с другом.



*Рис. 5.1***.** Различные графические примитивы диаграммы последовательности

Второе измерение диаграммы последовательности - вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем[[12]](#footnote-12). Сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа "раньше – позже". Возврат показывается пунктирной стрелкой, в обратном направлении.

***Линия жизни объекта.*** Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности.

Объект обязательно создается со своей линией жизни и, возможно, с фокусом управления.[[13]](#footnote-13)

***Фокус управления.*** В процессе функционирования систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия, или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название фокуса управления (focus of control, Activation Bar, Execution Occurances). Фокус управления изображается в форме вытянутого узкого прямоугольника (рис. 5.1), верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона - окончание фокуса управления (окончание активности).

Периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта имеются несколько фокусов управления (объект 4 на рис. 5.1).

Если объект инициирует рекурсивное взаимодействие с самим собой, то на диаграмме это обозначается небольшим прямоугольником, присоединенным к правой стороне фокуса управления такого объекта (объект 3 на рис.  5.1).

***1.2. Сообщения***

Взаимодействие объектов описывается совокупностью сообщений, которыми они обмениваются. Сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

Таким образом, сообщения не только передают некоторую информацию, но и требуют или предполагают от принимающего объекта выполнения ожидаемых действий. Сообщения могут инициировать выполнение операций объектом соответствующего класса, и параметры этих операций передаются вместе с сообщением. На диаграмме последовательности все сообщения упорядочены по времени своего возникновения в моделируемой системе.

Сообщения изображаются горизонтальными стрелками, соединяющими фокусы управления двух объектов. При этом неявно предполагается, что время передачи сообщения достаточно мало по сравнению с процессами выполнения действий объектами. Считается также, что за время передачи сообщения с соответствующими объектами не может произойти никаких событий.

Принято различать синхронное сообщение (synchronous message), когда посылающий его объект должен ждать пока обработка сообщения не будет завершена (например, вызов подпрограммы), и асинхронное сообщение (asynchronous message), когда посылающие его объект может продолжать работу и не должен ждать ответа. Синхронные сообщения изображаются стрелками с закрашенным концом, асинхронные – обычными стрелками.

Иногда отправителя сообщения называют клиентом, а получателя - сервером. При этом сообщение от клиента имеет форму запроса некоторого сервиса, а реакция сервера на запрос после получения сообщения может быть связана с выполнением определенных действий или передачи клиенту необходимой информации тоже в форме сообщения. Такие сообщения изображаются пунктирной стрелкой.

В отдельных случаях объект может посылать сообщения самому себе, инициируя так называемые рефлексивные сообщения. Такие сообщения изображаются прямоугольником со стрелкой. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента (объект 3 на рис. 5.1).

***Стереотипы сообщений.*** В языке UML предусмотрены некоторые стандартные действия, выполняемые в ответ на получение соответствующего сообщения. Они могут быть явно указаны на диаграмме последовательности в форме стереотипа рядом с сообщением, к которому они относятся. В этом случае они записываются в кавычках. Используются следующие обозначения для моделирования действий:

1) "call" (вызвать) - сообщение, требующее вызова операции или процедуры принимающего объекта. Если сообщение с этим стереотипом рефлексивное, то оно инициирует локальный вызов операции у самого пославшего это сообщение объекта;

2) "return" (возвратить) - сообщение, возвращающее значение выполненной операции или процедуры вызвавшему ее объекту.

3) "create" (создать) - сообщение, требующее создания другого объекта для выполнения определенных действий. Созданный объект может получить фокус управления, а может и не получить его;

4) "destroy" (уничтожить) - сообщение с явным требованием уничтожить соответствующий объект. Посылается в том случае, когда необходимо прекратить нежелательные действия со стороны существующего в системе объекта, либо когда объект больше не нужен и должен освободить задействованные им системные ресурсы;

4) "send" (послать) - обозначает посылку другому объекту некоторого сигнала, который асинхронно инициируется одним объектом и принимается (перехватывается) другим. Отличие сигнала от сообщения заключается в том, что сигнал должен быть явно описан в том классе, объект которого инициирует его передачу.

Кроме стереотипов, сообщения могут иметь собственное обозначение операции, вызов которой они инициируют у принимающего объекта. В этом случае рядом со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все равно должны присутствовать после имени операции. Примерами таких операций могут служить следующие: "выдать клиенту наличными сумму (п)", "установить соединение между абонентами (а, б)", "сделать вводимый текст невидимым ()", "подать звуковой сигнал тревоги ()".

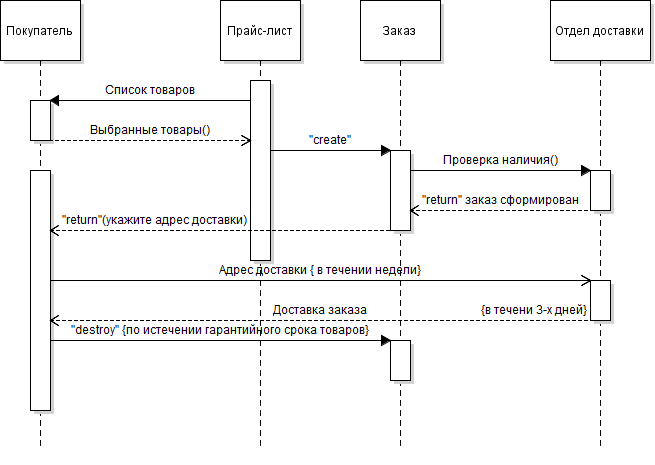
***Временные ограничения.*** В отдельных случаях выполнение тех или иных действий на диаграмме последовательности может потребовать явной спецификации временных ограничений, накладываемых на сам интервал выполнения операций или передачу сообщений. В языке UML для записи временных ограничений используются фигурные скобки. Временные ограничения могут относиться как к выполнению определенных действий объектами, так и к самим сообщениям, явно специфицируя условия их передачи или приема. Временные ограничения имеют обязательный или директивный характер для ассоциированных с ними объектов.

Временные ограничения могут записываться рядом с началом стрелки соответствующего сообщения. Если временная характеристика относится к конкретному объекту, то имя этого объекта записывается перед именем характеристики и отделяется от нее точкой.

***Комментарии или примечания.*** Комментарии или примечания уже рассматривались ранее при изучении других видов диаграмм. Комментарии, могут включаться и в диаграммы последовательности подобно другим диаграммам , ассоциируясь с отдельными объектами или фокусами управления. При этом используется стандартное обозначение для комментария - прямоугольник с "заломленным" правым верхним углом. Внутри этого прямоугольника записывается текст комментария на естественном языке.

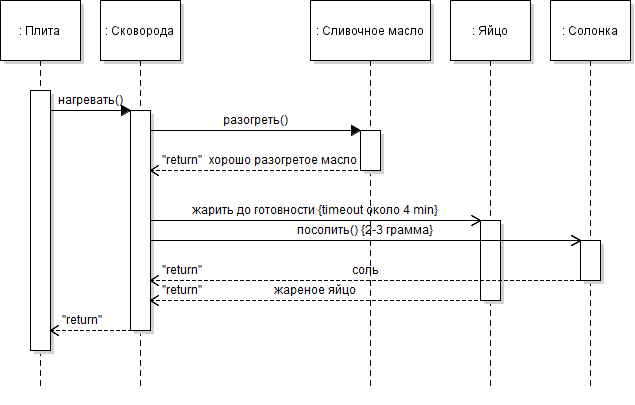
***2. Примеры диаграмм последовательности***

***1. Интернет – магазин***. На рис. 5.2 представлена диаграмма последовательности работы интернет – магазина. Покупателю предлагается выбрать товары из прайс-листа, выбранные товары образуют заказ, который проверяется на наличие в магазине указанных в нем товаров, затем определяется адрес доставки, и наконец, осуществляется доставка.



*Рис.5.2.* Диаграмма последовательности интернет - магазина

***2. Приготовление яичницы***. Традиционный учебный пример диаграммы последовательности, а именно, «процесс приготовление яичницы» приведен на рис. 5.3. ([*http://programador.ru/sequence-diagram*](http://programador.ru/sequence-diagram)) .



*Рис. 5.3*. Диаграмма последовательности процесса приготовления яичницы

На диаграмме указывается, что для приготовления яичницы необходимо:

* не забыть включить плиту:
* перед жаркой нужно хорошо разогреть масло;
* первым нужно положить масло, а потом яйца, а не наоборот;
* cоль можно добавить *асинхронно*, в процессе жарки.

***3. Рекомендации по построению диаграмм последовательности***

Построение диаграммы последовательности целесообразно начинать с выделения из всей совокупности тех и только тех классов, объекты которых участвуют в моделируемом взаимодействии. После этого все объекты наносятся на диаграмму с соблюдением некоторого порядка инициализации сообщений. Здесь необходимо установить, какие объекты будут существовать постоянно, а какие временно - только на период выполнения ими требуемых действий.

Когда объекты визуализированы, можно приступать к спецификации сообщений. При этом следует учитывать те роли, которые играют сообщения в системе. При необходимости уточнения этих ролей надо использовать их разновидности и стереотипы. Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение.

Детализация диаграммы последовательности связана с введением временных ограничений на выполнение отдельных действий в системе. Для простых асинхронных сообщений временные ограничения могут отсутствовать. Однако необходимость синхронизировать сложные потоки управления, как правило, требует введения в модель таких ограничений.

**Содержание отчета**

Диаграммы последовательности для разработанных ранее диаграмм вариантов использования (лабораторная работа № 1) и диаграмм классов (лабораторная работа № 2) и их количественные оценки.

**Контрольные вопросы**

1. Как на диаграмме последовательности представляется взаимодействие объектов?

2. Как указывается принадлежность объекта классу?

3. Какой символ определяет активность объекта?

4. В чем отличие реакции на синхронные и асинхронные сообщения?

5. Что отличает сигнал от сообщения и как они представляются на диаграмме?

**ЛАБОРАТОРНАЯ РАБОТА № 6**

**ДИАГРАММА ОБЪЕКТОВ**

**Цель работы**

Сформировать навыки анализа значений атрибутов объектов систем.

**Задание**

Разработать диаграммы объектов (object diagram) системы для диаграммы классов (лабораторная работа № 2) и написать комментарии к ней.

**Порядок выполнения**

***1. Описание диаграммы***

В отличие от диаграмм классов, диаграммы объектов не позволяют полностью описать объектную структуру системы. У класса может быть большое количество различных экземпляров, а при наличии нескольких классов, связанных друг с другом отношениями, число возможных конфигураций объектов многократно возрастает, и диаграмма объектов, таким образом, есть отображение множества объектов и отношений между ними лишь в некоторый момент времени.

### ***1.1. Объекты***

Объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он имеет свое собственное имя и конкретные значения атрибутов. В силу самых различных причин может возникнуть необходимость показать взаимосвязи не только между классами модели, но и между отдельными объектами, реализующими эти классы. Для этого используется диаграмма объектов.

Для графического изображения объектов используется такой же символ прямоугольника, что и для классов, в котором отсутствует секция методов. Имя объекта представляет собой строку текста, разделенную двоеточием:

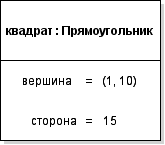
*<имя объекта>’ :’< Имя класса>*

Если имя объекта отсутствует, то предполагается, что объект является анонимным, на что указывает двоеточие. Отсутствие двоеточие означает, что в имени объекта отсутствует имя класса (рис. 6.1).[[14]](#footnote-14). В секции атрибутов записываются принимаемые ими значения в том или ином состоянии системы (рис. 6.2).



а б в

Рис. 6.1. Примеры имен объектов

На диаграмме объектов каждый объект представляет собой экземпляр соответствующего класса, а отношения между объектами описываются с помощью связей (links), которые являются экземплярами соответствующих отношений. Все связи изображаются сплошными линиями (рис. 6.3). Рис. 6.2 Пример объекта

***1.2. Использование диаграмм объектов***

Использование диаграмм объектов в прямом проектировании (создании кода на основе модели) невозможно, поскольку во время работы приложения экземпляры классов то создаются, то разрушаются. Диаграмма объектов представляет только одну из возможных структур, образуемых объектами, множество таких структур задается диаграммой классов. Однако весьма целесообразным является использование диаграмм объектов в обратном проектировании (создание модели на основе кода).

При обратном проектировании построение диаграммы объектов производится следующим образом.

1. Выбирается т.н. базовая точка реконструкции – какая-либо операция или экземпляр конкретного класса.

2. С помощью инструментальных средств или в соответствии со сценарием работы приложения фиксируется состояние системы в некоторый момент времени.

3. Идентифицируется множество объектов, сотрудничающих в данном контексте, и они изображаются на диаграмме объектов.

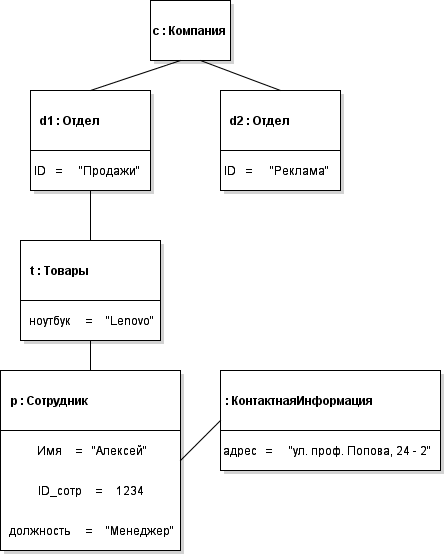
4. Чтобы обеспечить понимание семантики, идентифицируются связи, существующие между объектами.

5. Если диаграмма оказалась слишком сложной, из нее убираются объекты, не существенные для прояснения выбранного сценария.

6. Если диаграмма слишком проста, в нее добавляется окружение некоторых представляющих интерес объектов.

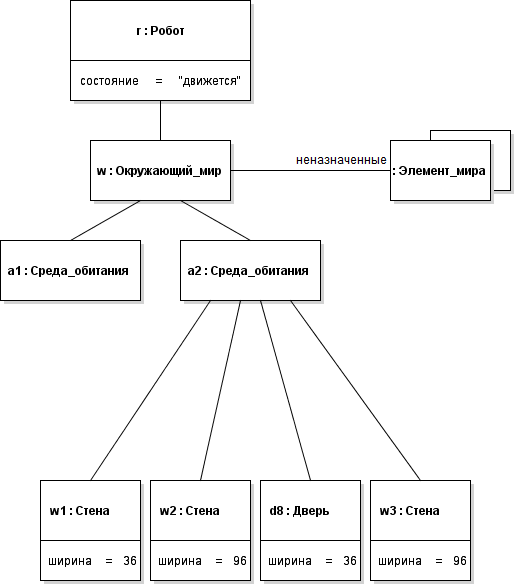
***2. Примеры диаграмм объектов***

На рис. 6.3 показано множество объектов (экземпляров классов) и отношений между ними в некоторый момент времени работы некоторого интернет – магазина, продающего компьютерную технику.



*Рис. 6.3*. Диаграмма объектов «Продажа ноутбука «Lenovo»

На рис. 6.4 показана совокупность объектов, взятая из реализации автономного робота. Внимание здесь акцентировано на нескольких объектах, составляющих часть механизма робота, предназначенного для расчета модели мира, в котором тот перемещается. Разумеется, в работе системы принимает участие гораздо больше объектов, но в этой диаграмме рассматриваются только абстракции, непосредственно вовлеченные в процесс формирования взгляда на мир.



*Рис. 6.4*. Диаграмма объектов «Движение робота по комнате

Как видно из рисунка, один из объектов соответствует самому роботу («r» - экземпляр класса «Робот»); в настоящий момент он находится в состоянии «движется». Этот объект связан с экземпляром «w» класса «Окружающий\_мир», являющегося абстракцией модели мира робота. В свою очередь объект «w» связан с мультиобъектом, который состоит из экземпляров класса «Элемент\_мира», описывающего сущности, опознанные роботом, но еще не включенные в его модель мира. Эти элементы помечены как части глобального состояния робота.

В текущий момент времени экземпляр «w» связан с двумя экземплярами класса «Среда\_Обитания». У одного из них («а2») показаны его собственные связи с тремя объектами класса «Стена» и одним – «Дверь», для которых указана их ширина. Как видно из диаграммы, робот распознал, что замкнутое помещение, в котором он находится, имеет с трех сторон стены, а с четвертой – дверь.

***3. Рекомендации по построению диаграмм объектов***

Диаграмма объектов на языке UML- это всего лишь графическое представление статического вида системы с точки зрения проектирования или процессов. Это означает, что ни одна отдельно взятая диаграмма объектов не в состоянии передать всю заключенную в этих видах информацию. На самом деле во всех системах, кроме самых тривиальных, существуют сотни, а то и тысячи объектов, большая часть которых анонимна. Полностью специфицировать все объекты системы и все способы, которыми они могут быть ассоциированы, невозможно. Следовательно, диаграммы объектов должны отражать только некоторые конкретные объекты или прототипы, входящие в состав работающей системы.

Хорошо структурированная диаграмма объектов характеризуется следующими свойствами:

1) отражает только один аспект статического вида системы с точки зрения проектирования или процессов;

2) содержит только существенные для понимания данного аспекта элементы;

3) уровень детализации соответствует уровню абстракции системы. (показаны только те значения атрибутов и дополнения, которые существенны для понимания).

Изображая диаграмму объектов, следует придерживаться следующих правил:

1) элементы должны быть расположены так, чтобы число пересечений было минимальным;

2) семантически близкие сущности должны быть расположены рядом;

3) важные особенности диаграммы должны быть снабжены примечаниями.

**Содержание отчета**

Диаграммы объектов для разработанной ранее диаграммы классов (лабораторная работа № 2), и комментарии к ним.

**Контрольные вопросы**

1. Какая секция символа класса диаграммы классов отсутствует в символе объекта диаграммы объектов?

2. Когда в имени объекта необходимо использовать двоеточие?

3. Какие типы отношений между классами могут быть представлены на диаграмме объектов?

4. Сколько диаграмм объектов может быть построено для одной диаграммы классов?

**ЛАБОРАТОРНАЯ РАБОТА № 7**

**ДИАГРАММА EPC**

**Цель работы**

Сформировать навыки разработки событийной модели программной поддержки бизнес-процессов.

**Задание**

Разработать диаграммы EPC (*event-driven process chain*) бизнес-процессов системы, для которой заданы диаграммы состояний (лабораторная работа № 3) и диаграммы деятельности (лабораторная работа № 4), и написать комментарии к ним.

**Порядок выполнения**

***1. Описание диаграммы***

Нотация EPC (Event-Driven Process Chain – событийная цепочка процессов) используется для описания процессов нижнего уровня. Диаграмма процесса в нотации EPC, представляет собой упорядоченную комбинацию событий и функций. Для каждой функции могут быть определены начальные и конечные события, участники, исполнители, материальные и документальные потоки, сопровождающие её, а также проведена декомпозиция на более низкие уровни. Декомпозиция может производиться в нотациях EPC или BPMN.

***1.1. Нотация EPC***

Описание нотации EPC, сведено в таблицу 7.1.

*Таблица 7.1*

**Назначение графических символов диаграмм EPC**

| Название | Графический символ | Описание |
| --- | --- | --- |
| Связь (Connection) | T_1-1.jpeg | Стрелка отображает связи элементов диаграммы процесса EPC между собой |
| Событие (Event) | T_1-2.jpeg | Событие – состояние, которое является существенным для целей управления бизнесом и оказывает влияние или контролирует дальнейшее развитие одного или более бизнес-процессов. Элемент отображает события, активизирующие функции или порождаемые функциями. Внутри блока помещается наименование события |

*Продолжение табл. 7.1*

| Название | Графический символ | Описание |
| --- | --- | --- |
| Процесс (функция, Activity | T_1-3.jpeg | Блок представляет собой функцию – процесс, действие или набор действий, выполняемых над исходным объектом (документом, товарно-материальными ценностями (ТМЦ) и прочим) с целью получения заданного результата. Внутри блока помещается наименование функции. Временная последовательность выполнения функций задается расположением функций на диаграмме процесса сверху вниз. |
| Оператор AND (AND rule, «И») | T_1-5.jpeg | Оператор "И" используется для обозначения слияния / ветвления как функций, так и событий. Если завершение выполнения функции должно инициировать одновременно несколько событий, то это обозначается с помощью оператора "И", следующего после функции и перед событиями (рис. 7.1а). Если событие происходит только после обязательного завершения выполнения нескольких функций, то это обозначается с помощью оператора "И", следующего после функций и перед одиночным событием (рис. 7.1б). Если функция может начать выполняться только после того, как произойдут несколько событий, то это обозначается с помощью оператора "И", следующего после событий и перед функцией (рис. 7.1в). Если одно событие может инициировать одновременное выполнение нескольких функций, то это обозначается с помощью оператора "И", следующего после события и перед функциями (рис. 7.1г). |

*Продолжение табл. 7.1*

| Название | Графический символ | Описание |
| --- | --- | --- |
| Оператор XOR (XOR rule, "Исключающее ИЛИ") | T_1-4.jpeg | Оператор "Исключающее ИЛИ" используется для обозначения слияния/ветвления функций и для слияния событий. По правилам нотации EPC после одиночного события не может следовать разветвляющий оператор "Исключающее ИЛИ". Если завершение выполнения функции может инициировать только одно из событий в зависимости от условия, то это обозначается с помощью оператора "Исключающее ИЛИ", следующего за функцией и перед событиями (рис. 7.2а). Если событие происходит сразу после завершения выполнения либо одной функции, либо другой, то это обозначается с помощью оператора "Исключающее ИЛИ", следующего после функций и перед одиночным событием (рис. 7.2б). Если функция может начать выполняться сразу после того, как произойдет либо одно событие, либо другое, то это обозначается с помощью оператора "Исключающее ИЛИ", следующего после нескольких событий и перед функцией (рис. 7.3в). |
| Оператор OR (OR rule, "ИЛИ") | T_1-6.jpeg | Оператор "ИЛИ" используется для обозначения слияния/ветвления функций и для слияния событий. По правилам нотации EPC после одиночного события не может следовать разветвляющий оператор "ИЛИ". Если завершение выполнения функции может инициировать одно или несколько событий, то это обозначается с помощью оператора "ИЛИ", следующего после функции и перед событиями (рис. 7.3а). Если событие происходит после завершения выполнения одной или нескольких функций, то это обозначается с помощью оператора "ИЛИ", следующего после функций и перед одиночным событием (рис. 7.3б). Если функция может начать выполняться после того, как произойдет одно или несколько событий, то это обозначается с помощью оператора "ИЛИ", следующего после событий и перед функцией (рис. 7.3в) |
| Организа-ционная единица (Organizational unit) | T_1-7.jpeg | Используется для отображения на диаграмме организационных единиц (отдела, бюро, подразделения, кафедры) – участников функций. Внутри блока помещается наименование организационной единицы. |

*Продолжение табл. 7.1*

| Название | Графический символ | Описание |
| --- | --- | --- |
| Должность (Role) | T_1-8.jpeg | Используется для отображения на диаграмме должностей (ролей) – исполнителей или участников функций. Внутри блока помещается наименование должности. |
| Исполнитель (Person) | T_1-9.jpeg | Используется для отображения на диаграмме исполнителей, владельцев или участников функций. Внутри блока помещается наименование исполнителя. |
| Местоположение (Location) | T_1-10.jpeg | Местоположение, расположение, положение, место, местонахождение, нахождение, участок |
| Предприятие (Entity) | T_1-11.jpeg | Используется для отображения на диаграмме внешних исполнителей или участников функций. Внутри блока помещается наименование предприятия. |
| База данных (Database) | T_1-12.jpeg | Используется для отображения на диаграмме базы данных, сопровождающей выполнение функции. Внутри блока помещается наименование базы данных. |
| Документ (Docu-ment) | T_1-13.jpeg | Используется для отображения на диаграмме бумажных документов и (или) электронных документов, сопровождающих выполнение функции. Внутри блока помещается наименование документа. |
| ИТ система (IT system) | T_1-14.jpeg | Используется для отображения на диаграмме информационной системы, поддерживающей выполнение функции. Внутри блока помещается наименование информационной системы. |
| ТМЦ (Product) | T_1-15.jpeg | Используется для отображения на диаграмме товарно-материальных ценностей (ТМЦ), сопровождающих выполнение функции. Внутри блока помещается наименование ТМЦ |

*Окончание табл. 7.1*

| Название | Графический символ | Описание |
| --- | --- | --- |
| Риск (Risk) | T_1-16.jpeg | Обозначает опасность, угрожающую выполнению фунций. Внутри блока помещается наименование риска. |
| Интерфейс процесса (Process interface) | T_1-17.jpeg | Элемент, обозначающий внешний (по отношению к текущей диаграмме) процесс или функцию. Используется для указания взаимосвязи процессов - обозначает предыдущий или следующий процесс по отношению к диаграмме рассматриваемого процесса (рис. 7.4*а*); - обозначает процесс, откуда поступил или куда передается объект (рис. 7.4*б,в*). Внутри блока помещается наименование внешнего процесса. |

Примеры использования символов нотации ARIS приведены на Рис. 7.1, - Рис. 7.6.

|  |  |
| --- | --- |
| Рис_1а.jpeg  *а* | Рис_1б.jpeg  *б* |
|  |  |
| Рис_1в.jpeg  *в* | Рис_1г.jpeg  *г* |

*Рис. 7.1*. Примеры использования оператор «AND»: *а* - завершение выполнения процесса "Функция" одновременно инициирует события: "Событие1" и "Событие2";*б* - событие "Событие" произойдет только после обязательного завершения процессов "Функция 1" и "Функция 2"; *в* - процесс "Функция" начнет выполняться только после того, как произойдут событие "Событие 1" и событие "Событие 2"; *г* - событие "Событие" одновременно инициирует выполнение процессов "Функция 1" и "Функция 2".

|  |  |  |
| --- | --- | --- |
| Рис_2а.jpeg*а* | Рис_2б.jpeg*б* | Рис_2в.jpeg*в* |

*Рис. 7.2*. Примеры использования оператор «XOR»: *а* - процесс "Функция" инициирует либо только событие "Событие 1", либо только событие "Событие 2"; *б* - событие "Событие" может произойти либо сразу после завершения выполнения процесса "Функция 1", либо сразу после завершения выполнения процесса "Функция 2"; *в* - процесс "Функция" может начать выполняться сразу после того, как произойдет либо событие "Событие 1", либо событие "Событие 2".

|  |  |  |
| --- | --- | --- |
| Рис_3а.jpeg*а* | Рис_3б.jpeg*б* | Рис_3в.jpeg*в* |

*Рис. 7.3*. Примеры использования оператор «OR»: *а* - завершение выполнения процесса "Функция" может инициировать 3 вида ситуаций: либо возникнет только событие "Событие 1", либо возникнет только событие "Событие 2", либо возникнут одновременно и событие "Событие 1", и событие "Событие 2"; *б* - событие "Событие" может произойти либо после завершения выполнения процесса "Функция 1", либо после завершения выполнения процесса "Функция 2", либо после завершения выполнения и процесса "Функция 1", и процесса "Функция 2"; *в* - процесс "Функция" может начать выполняться либо после того, как произойдет событие "Событие 1", либо после того, как произойдет событие "Событие 2", либо после того, как произойдут оба события: "Событие 1" и "Событие 2".

|  |  |
| --- | --- |
| Рис_4а.jpeg  *а* | |
|  | |
| Рис_4б.jpeg  *б* | Рис_4в.jpeg  *в* |

*Рис. 7.4*. Примеры использования символа «Process interface»: *а* - договор является результатом выполнения процесса "Заключение договора"; *б* - после окончания процесса "Процесс 1" и наступления события "Событие 1" начинает выполняться процесс "Процесс 2"; *в* - перед началом процесса "Процесс 2" был завершен процесс "Процесс 1", инициировавший событие "Событие 1"

|  |  |
| --- | --- |
| Рис_5а.jpeg  *а* | Рис_5б.jpeg  *б* |

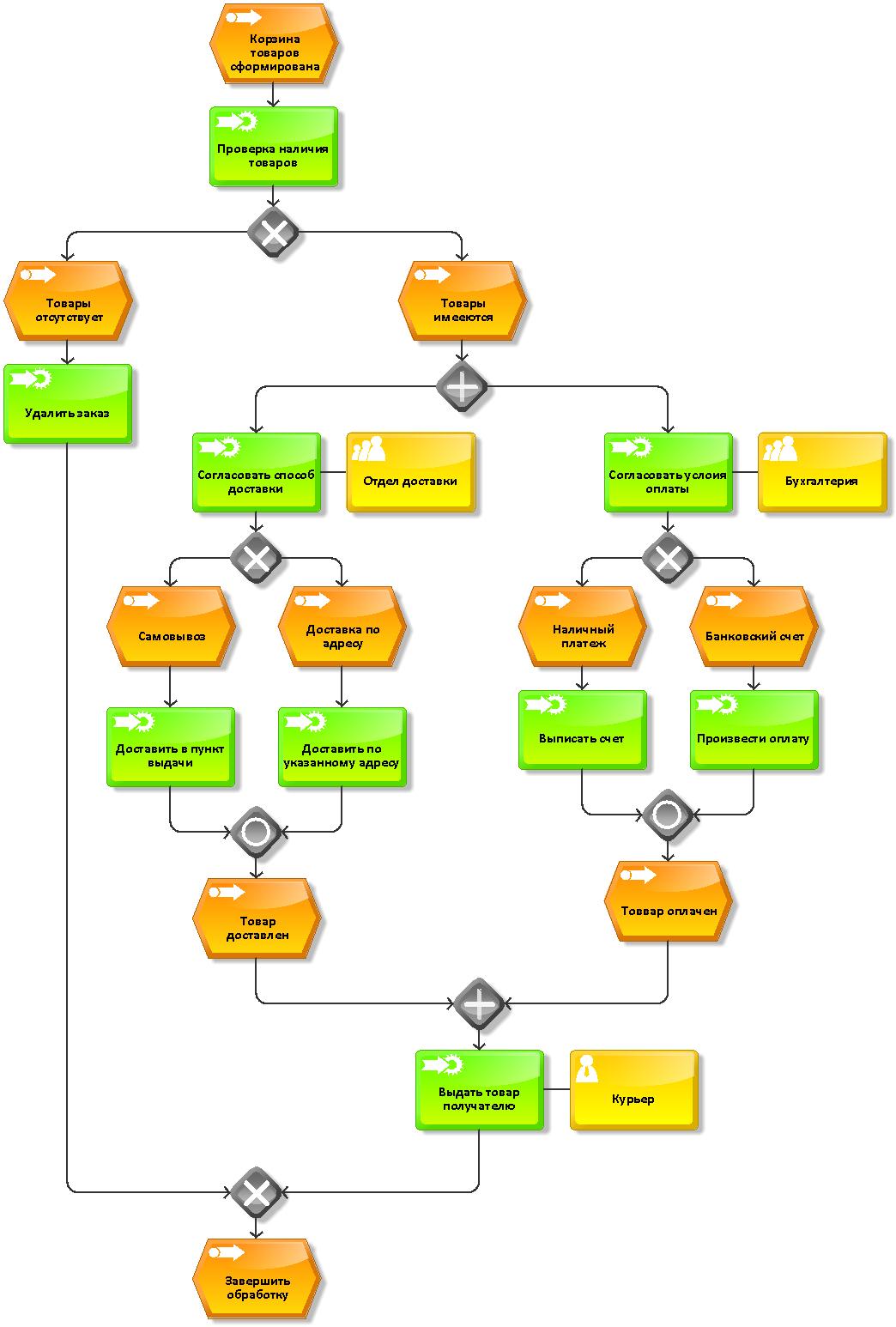
*Рис. 7.5*. Пример декомпозиции диаграмм EPC: *а* – исходная диаграмма; *б* – диаграмма декомпозиции процесса «Функция 1».

|  |  |  |
| --- | --- | --- |
| Рис_6а.jpeg*а* | Рис_6б.jpeg*б* | Рис_6в.jpeg*в* |
|  |  |  |
| Рис_6г.jpeg  *г* | Рис_7.jpeg  *д* | |

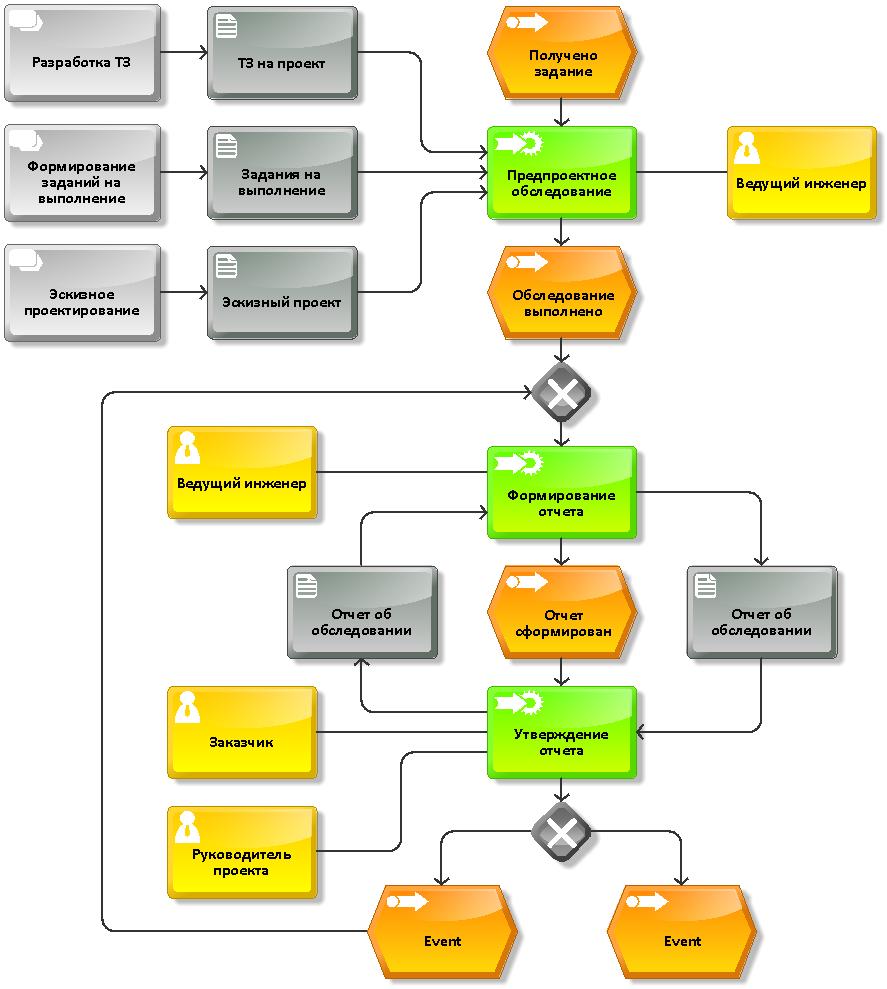
*Рис. 7.6*. Примеры использования логических символов: *а,б, в, г* – допустимые комбинации; *д-* не допустимая комбинация

### ***1.2. Примеры диаграмм EPC***

Примеры диаграмм EPC приведены на Рис. 7.7 и Рис. 7.8.



*Рис. 7.7*. Процесс обслуживания покупателя в интернет-магазине

**

*Рис. 7.8.*EPC диаграмма предпроектного обследования предприятия

### ***2. Рекомендации по построению диаграмм EPC***

1. Диаграмма функции EPC должна начинаться как минимум одним стартовым событием (стартовое событие может следовать за интерфейсом процесса) и завершаться как минимум одним конечным событием (конечное событие может предшествовать интерфейсу процесса).

2. События и функции по ходу выполнения процесса должны чередоваться. Решения о дальнейшем ходе выполнения процесса принимаются функциями.

3. Рекомендуемое количество функций на диаграмме – не более 20. Если количество функций диаграммы значительно превышает 20, то существует вероятность, что неправильно выделены процессы на верхнем уровне и необходимо произвести корректировку модели.

4. События и функции должны содержать строго по одной входящей и одной исходящей связи, отражающей ход выполнения процесса.

5. События и операторы, окружавшие функцию на декомпозированной диаграмме (Рис. 7.5*а*), должны быть начальными (результирующими) событиями и операторами на диаграмме результата декомпозиции (Рис. 7.5б).

6. На диаграмме не должны присутствовать объекты без единой связи.

7. Каждый оператор слияния должен обладать хотя бы двумя входящими связями и только одной исходящей, оператор ветвления – только одной входящей связью и хотя бы двумя исходящими. Операторы не могут обладать одновременно несколькими входящими и исходящими связями.

8. Если оператор обладает входящей связью от элемента "событие", то он должен обладать исходящей связью к элементу "функция" и наоборот.

9. За одиночным событием не должны следовать операторы "OR (ИЛИ)" или "XOR (Исключающее ИЛИ)".

10. Операторы могут объединять или разветвлять только функции или только события. Одновременное объединение/ветвление функции и события невозможно.

11. Оператор, разветвляющий ветки, и оператор, объединяющий эти ветки, должны совпадать. Допускается также ситуация, когда оператор ветвления "AND", оператор объединения – "OR" (рис. 7.6).

**Содержание отчета**

Диаграммы EPC и комментарии к ним.

**Контрольные вопросы**

1. Какими символами в нотации EPC описывается внешнее окружение моделируемой системы?

2. Какие символы используются для задания альтернативных потоков управления?

3. Каким образом в нотации задается параллельное выполнение процессов?

4. Каким свойством должны обладать события, непосредственно следующие после символа «XOR»?

5. Какие символы повторяются на диаграммах процесса до его декомпозиции и после?

**ЛАБОРАТОРНАЯ РАБОТА № 8.**

**ДИАГРАММА BPMN**

**Цель работы**

Сформировать навыки разработки функциональной модели выполнения бизнес-процессов.

**Задание**

Разработать диаграммы BPMN (*Business Process Model and Notation*) бизнес-процессов системы, для которой заданы диаграммы вариантов использования (лабораторная работа № 1), диаграммы состояний (лабораторная работа № 3) и диаграммы деятельности (лабораторная работа № 4), и написать комментарии к ним.

**Порядок выполнения**

***1. Описание диаграммы***

Нотация BPMN (Business Process Model and Notation – модель бизнес-процессов и нотация) используется для описания процессов нижнего уровня. Диаграмма процесса в нотации BPMN представляет собой алгоритм выполнения процесса. На диаграмме могут быть определены события, исполнители, материальные и документальные потоки, сопровождающие выполнение процесса. Каждый процесс может быть декомпозирован на более низкие уровни. Декомпозиция может производиться в нотациях BPMN или EPC.

В нотации BPMN выделяют пять основных категорий элементов:

1) соединяющие элементы;

2) зоны ответственности (пулы и дорожки);

3) объекты потока управления (события, процессы и шлюзы);

4) данные (объекты данных и базы данных);

5) артефакты.

Описание назначения графических символов, используемых в нотации BPMN, приведены в таблицах 8.1–8.5 и на рисунках 8.1 - 8.7.

***1.1. Соединяющие элементы*** подразделяются на потоки управления, потоки сообщений и ассоциации (Таблица 8.1).

*Таблица 8.1.*

**Соединяющие элементы**[[15]](#footnote-15)

|  |  |  |
| --- | --- | --- |
| Название | Графический символ | Описание |
| Поток операций, поток управления(Sequence Flow) | Рис_8-1аI.jpg | Задает последовательность (до-после) возникновения событий и выполнения действий. |
| Поток сообщений(Message Flow) | Рис_8-1бI.jpg | Показывает сообщения, которыми обмениваются участники бизнес-процесса. |
| Ассоциация (Association) | Рис_8-1в.jpeg  Рис_8-1в2д.jpeg | Отражает связь между данными (артефактами) и объектами потока |
| Поток операций по умолчанию (Default Sequence Flow). | Рис_8-1г.jpeg | Поток считается верным в процессе, активизируется если активация других потоков в соответствии с их логическими выражениями или событиями невозможна |
| Условный поток операций (Conditional Sequence Flow). | Рис_8-1дI.jpg | Используется при ветвлении потоков. Условие активации потока задается рядом в виде логического выражения |

***1.2. Зоны ответственности*** обозначаются т.н. пулами (Pool, Participant) и дорожками (Lane). Пул предназначен для отображения потока рассматриваемого процесса. Содержимое пула представляет собой диаграмму процесса, который рассматривается. На диаграмме развернутый пул может быть только один. Дорожки предназначены для отображения организационных единиц (должности, подразделения, роли, внешнего субъекта) – исполнителей задач и подпроцессов процесса BPMN. Внутри блока помещается наименование организационной единицы (Рис. 8.1).

***1.3. Объекты потока управления*** подразделяются на три основных типа: события (events) – табл. 8.2, процессы (activities) - табл. 8.3) и логические операторы (gateways) – табл. 8.4.

***1.3.1. Событие*** - это состояние, которое является существенным для целей управления и оказывает влияние или контролирует дальнейшее развитие одного или более процессов.

|  |  |  |  |
| --- | --- | --- | --- |
| **Пул** | **Дорожка** | **Дорожка** |  |
| **Дорожка** |  |
| **Дорожка** |  |  |

*Рис. 8.1.* Пул и дорожки

Все события классифицируются по следующим признакам.

**I**. По времени наступления.

1. Cтартовое событие (Start event) инициирует начало процесса (диаграммы). Из стартового события поток управления может только исходить, а поток сообщений – как входить, так и исходить. На диаграмме процесса, как правило, отображается только одно стартовое событие, но оно может отсутствовать, или их может быть несколько при отображении процесса с пулами, дорожками или развернутыми подпроцессами.

2. Конечное событие (End event) – является результатом выполнения процесса. В конечное событие поток управления может только входить, а поток сообщений – как входить, так и исходить. В конечное событие может только входить поток (стрелка). На диаграмме конечное событие, как и стартовое, может быть одно, несколько (даже при отсутствии пулов и дорожек) или ни одного.

3. Промежуточное событие (Intermediate event) – все остальные события, возникающие в ходе выполнения процесса. В промежуточное событие обязательно должен входить и выходить один поток. Исключение составляет граничные (Boundary) события, возникающие и обрабатываемые непосредственно либо в самом начале действия либо в его конце. Такие события отображаются на границе (контуре) действия и у них может быть только либо входящий, либо исходящий поток.

**II.** По типу результата действия.

1. Событие-инициатор обработки (Catching) – стартовое или промежуточное событие, возникшее в результате выполнения действия и требующее его последующей обработки. Отображается не закрашенной иконкой;

2. Событие-результат обработки (Throwing)– промежуточное или конечное событие, возникшее в результате выполнения действия и являющееся итоговым результатом стандартного или нестандартного выполнения процесса. Отображается закрашенной иконкой.

**III**. По возможности прерывания выполнения действия (подпроцесса):

1. Не прерывающее событие – стартовое или промежуточное событие, возникающее в ходе выполнения действия, но инициирующее связанный с событием исходящий поток только после завершения действия. Контур события отображается штриховой линией;

2. Прерывающее событие – событие, возникающее до или после стандартного выполнения действия или требующее его немедленного прекращения в исключительных ситуациях. Например, при отсутствии всей необходимой информации или возникновении ошибки в ходе ее обработки, необходимости выполнения дополнительных действий и т.д. Контур события отображается сплошной линией;

Символы событий, обозначающие причину их возникновения приведены в табл. 8.2.

*Таблица 8.2*

**Символы событий**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Событие | | | | Описание |
| Стартовое событие (Start event) | Промежуточное событие (Intermediate event) | | Конечное событие (End event) |
| Инициатор обработки (Catching) | Результат обработки (Throwing) |
| Не определенное (None) | S_1-1.jpeg | ***S_1-2.jpeg*** |  | S_1-4.jpeg | Не типизированное событие. Используется для отображения начала или окончания процесса. |
| Сообщение (Message) | S_2-1.jpeg | S_2-2.jpeg | S_2-3.jpeg | S_2-4.jpeg | Показывает получение и отправку сообщений в ходе выполнения процесса. |
| Таймер (Timer) | S_3-1.jpeg | S_3-2.jpeg |  |  | Моделирует события, происходящие по расписанию (в определенные моменты или периоды времени). Позволяет моделировать таймауты (перерывы в ходе выполнения процесса). |

*Продолжение табл. 8.2*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Событие | | | | Описание |
| Стартовое событие (Start event) | Промежуточное событие (Intermediate event) | | Конечное событие (End event) |
| Инициатор обработки (Catching) | Результат обработки (Throwing) |
| Ошибка (Error) |  |  |  | S_4-4.jpeg | Моделирует генерацию и обработку ошибок (ошибки могут иметь различные типы). |
| Прерывание, эска-лация (Escalation) |  |  | S_5-3.jpeg | S_5-4.jpeg | Отражает факт возникновения и/или обработки ситуации, требующей немедленной реакции (более общая ситуация, чем ошибка, т.к. может привести к положительному завершению процесса). |
| Компенса-ция (Compensation) |  |  | S_6-3.jpeg | S_6-4.jpeg | Инициирует вспомогательные действия, компенсирующие неудачное завершение (прерывание) процесса. |
| Условие (Conditional) | S_7-1.jpeg | S_7-2.jpeg |  |  | Показывает получение и отправку сообщений в ходе выполнения процесса, тем самым интегрируя бизнес-правила в процесс. |
| Связь (Link) |  | S_8-2.jpeg | S_8-3.jpeg |  | Используется для связки процессов или частей. Может использоваться для межстраничного соединения. |
| Сигнал (Signal) | S_9-1.jpeg | S_9-2.jpeg | S_9-3.jpeg | S_9-4.jpeg | Отражает факт рассылки или приема сигналов несколькими процессами. Один сигнал может обрабатываться несколькими получателями. В отличие от сообщения не содержит информации. |

*Окончание табл. 8.2*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Название | Событие | | | | Описание |
| Стартовое событие (Start event) | Промежуточное событие (Intermediate event) | | Конечное событие (End event) |
| Инициатор обработки (Catching) | Результат обработки (Throwing) |
| Множест-венное (Multiple) | S_11-1.jpeg | S_11-2.jpeg | S_11-3.jpeg | S_11-4.jpeg | Отражает факт возникновения одного события из некоторого множества. |
| Параллель-но множест-венное (Parallel Multiple) | S_12-1.jpeg | S_12-2.jpeg |  |  | Отражает факт возникновения всех событий из некоторого множества. |
| Завершение (Terminate) |  |  |  | S_10-4.jpeg | Отражает факт немедленного завершения всего процесса |

***1.3.2. Процесс*** – это действие или набор действий, выполняемых над исходным объектом деятельности (документом, ТМЦ и прочим) с целью получения заданного результата.

Процессы BPMN подразделяются на задачи (Task) и подпроцессы (SubProcess).

***1.3.2.1 Задача*** – это простое действие (или операция), которое не имеет дальнейшей декомпозиции в рамках рассматриваемого процесса. Задачи подразделяются на типы, каждый из которых (за исключением абстрактной задачи) обозначается своим маркером в левом верхнем углу блока задачи, а внутрь помещается соответствующее наименование. Символы задач представлены в табл. 8.3.

*Таблица 8.3*

**Символы задач**

|  |  |  |
| --- | --- | --- |
| Название | Символ | Описание |
| Абстрактная задача | T_1.jpeg | Задача с не определенным типом |

*Окончание табл. 8.3*

|  |  |  |
| --- | --- | --- |
| Название | Символ | Описание |
| Бизнес-правило (Business rule task) | T_2.jpeg | Задача, суть которой заключается в выполнении бизнес-правила |
| Ручное выполнение (Manual task) | T_3.jpeg | Задача, выполнение которой подразумевает действия человека и исключает использование каких-либо автоматизированных механизмов исполнения или приложений |
| Получение сообщений (Receive task) | T_4.jpeg | Задача, суть которой заключается в получении сообщения от внешнего участника, находящегося за пределами рассматриваемого процесса |
| Задача-сценарий (Script task) | T_5.jpeg | Задача, суть которой заключается в выполнении некоторого сценария (или скрипта) – некоторой автоматической операции |
| Отправка сообщений (Send task) | T_6.jpeg | Задача, суть которой заключается в отправлении сообщения внешнему участнику за пределы рассматриваемого процесса |
| Сервисная задача (Service task) | T_7.jpeg | Задача, предназначенная для оказания услуги, которая может являться как web-сервисом, так и автоматизированным приложением |
| Пользовательская задача (User task) | T_8.jpeg | Задача, которую выполняет человек при содействии других людей или программного обеспечения |

***1.3.2.2. Подпроцесс*** – это декомпозированный процесс, включенный в состав рассматриваемого процесса и описанный более подробно на своей диаграмме. Принято различать развёрнутые подпроцессы (*expanded subprocess*), содержащие внутри себя собственную диаграмму бизнес-процессов, и свёрнутые подпроцессы (*collapsed subprocess*), в которых скрыты детали реализации. Символы развернутых типов подпроцессов представлены в табл. 8.4. Символы свернутых подпроцессов помечаются символом «+» на нижней части рамки (Рис. 8.2*б*).

*Таблица 8.4*

**Символы подпроцессов**

|  |  |  |
| --- | --- | --- |
| Название | Символ | Описание |
| Подпроцесс (Subprocess) | Pp_1.jpeg | Подпроцесс с неопределенным типом |
| Спонтанный подпроцесс (Ad-Hoc subprocess) | Pp_2.jpeg | Подпроцесс, представляющий собой группу процессов, взаимодействие между которыми не поддаются строго регламентированным правилам. Определяется только набор процессов, однако, их последовательность и количество выполнений определяются исполнителями этих процессов |
| Свернутый подпроцесс (Collapsed) | Pp_5.jpeg | Подпроцесс, для которого скрыты детали реализации |
| Событийный подпроцесс (Event subprocess) | Pp_3.jpeg | Подпроцесс, не имеющий входящих и исходящих потоков управления, запускается всякий раз, когда его стартовое событие запускается во время выполнения родительского процесса |

*Окончание табл. 8.4*

|  |  |  |
| --- | --- | --- |
| Название | Символ | Описание |
| Транзакция (Transaction) | Pp_4.jpeg | Подпроцесс, состоящий из набора процессов, которые в совокупности представляют некий неделимый процесс: либо весь процесс выполняется полностью, либо не выполняется вообще. Транзакции используются тогда, когда необходимо выполнить несколько процессов, но при каких-то исключительных ситуациях необходимо «откатить» выполняемые процессы |

Если диаграмма содержит одинаковые фрагменты, то они могут быть описаны отдельной диаграммой, обращение к которой представляется символом «Call activity» (Вызов процесса). Этот символ отличается от символа «Task» (Задача) жирной рамкой (Рис. 8.2). Вызываемый процесс в этом случае является глобальным по отношению к процессу, представленному на диаграмме, из которой он вызывается.

|  |  |
| --- | --- |
| Ca.jpeg | Pp_6.jpeg |
| *а* | *б* |

*Рис. 8.2.*Вызов процесса: *а* - развёрнутый подпроцесс (*expanded subprocess*); *б* - свёрнутый подпроцесс (*collapsed subprocess*)

***1.3.3. Шлюзы*** (Gateway) используются для контроля расхождений и схождений потока операций. Таким образом, данный термин подразумевает ветвление, раздвоение, слияние и соединение маршрутов. Внутренние маркеры указывают тип контроля развития процесса. Символы шлюзов представлены в таблице 8.5.

*Таблица 8.5*

**Символы шлюзов**

|  |  |  |
| --- | --- | --- |
| Название | Символ | Описание |
| Неопределенный шлюз (Gateway) | G1.jpeg | При разделении направляет поток в одну исходящую ветвь. При объединении ожидает завершения одной входящей ветви перед инициированием исходящего потока. |

*Окончание табл. 8.5*

|  |  |  |
| --- | --- | --- |
| Название | Символ | Описание |
| Параллельный шлюз (AND, "И") Parallel gateway | G2.jpeg | Предназначен для слияния/ветвления одновременно (параллельно) выполняемых потоков операций. |
| Эксклюзивный шлюз (XOR, "Исключающее ИЛИ") Exclusive gateway | G3.jpeg | Предназначен для разделения потока операций на несколько альтернативных маршрутов, когда в ходе выполнения процесса может быть активирован только один из предложенных маршрутов. Условия пропуска по исходящему маршруту задается рядом с соответствующей линией в виде логического выражения |
| Эксклюзивный шлюз по событиям (XOR, "Исключающее ИЛИ") Event-based gateway | G4.jpeg | Предназначен для разделения потока операций на несколько альтернативных маршрутов. Единственный маршрут, по которому будет продолжен процесс, выбирается не на основе логического выражения, а в зависимости от произошедших событий, которые указываются по соответствующему маршруту. |
| Эксклюзивный, основанный на событиях, шлюз Instantiating event-based gateway | G5.jpeg | Аналогичен предыдущему шлюзу (Event-based gateway), но используется в качестве начального символа процесса (подпроцесса). Не имеет входящих потоков. |
| Неэксклюзивный шлюз (OR, "ИЛИ") Inclusive gateway | G6.jpeg | Предназначен для разделения потока операций на несколько маршрутов, каждый из которых активируется при условии истинности связанного с ним логического выражения. Таким образом, при выполнении процесса может быть выбрано сразу несколько маршрутов, в т.ч. и ни одного в случае ложности всех выражений. |
| Комплексный шлюз  Complex gateway | G7.jpeg | Аналогичен предыдущему шлюзу (Inclusive gateway) шлюзу. Отличие заключается в том, что с ним связано одно выражение, которое определяет, какие из потоков операций будут активированы |
| Параллельный, основанный на событиях шлюз Instantiating parallel event-based gateway | G8.jpeg | Аналогичен шлюзу Instantiating event-based gateway, но возможна активация сразу нескольких маршрутов в случае срабатывания событий, с которыми они связаны. Возможно асинхронное выполнение маршрутов (связанных потоков операций и действий), когда после активации и начала выполнения одного из маршрутов, другие маршруты тоже могут быть активированы и выполнены, пока не наступил момент завершения процесса (подпроцесса). Не имеет входящих потоков. |

***1.4. Данные.*** В спецификации BPMN данные не относятся к Элементам потока, поскольку они не оказывают непосредственного влияния на Поток операций или Поток сообщений, присутствующих в Процессе, однако, содержат сведения о данном Процессе, заключающиеся в описании того, какие документы, сведения или какие-либо другие объекты используются и дополняются в ходе выполнения Процесса.

***1.4.1. Объекты данных***. Символы Объектов данных (Data object) отображают данные, сопровождающие выполнение процесса. В качестве объекта данных может использоваться объект любого из следующих справочников: Бумажный документ, Электронный документ, ТМЦ, Информация, Программные продукты, Термины, Прочее. Символы объектов данных представлены на рис. 8.3.

|  |  |  |
| --- | --- | --- |
| D1.jpeg  *а* | D2.jpeg  *б* | D3.jpeg  *в* |

*Рис. 8.3.* Символы объектов данных: *а* - неопределенные данные; *б* - входные данные (Data Input); *в* - выходные данные (Data output)

***1.4.2. Базы данных***. Символ Базы данных (Data store) используется для отображения на диаграмме базы данных, сопровождающей выполнение процесса (рис. 8.4).



*Рис. 8.4.* Символ Базы данных

***1.5. Артефакты.*** Дополнительная информация о Процессе, не связанная с потоками его операций и/или сообщений образует 3 типа артефактов диаграмм BPMN:

1) сообщение (Message);

2) текстовая аннотация (Text annotation)

3) группа (Group).

Сообщение используется для отображения сущности взаимодействия между двумя Участниками процесса (Рис. 8.5*а*).

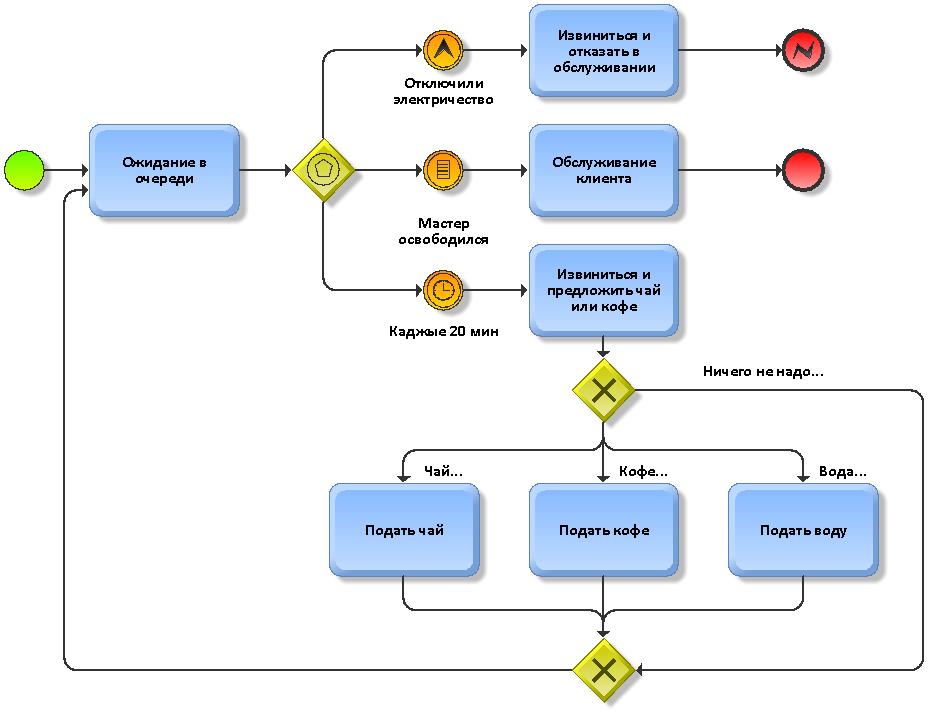
Текстовая Аннотация (Рис. 8.5*б*) присоединяется к определенному элементу на диаграмме при помощи Ассоциации (Табл. 8.1), и не оказывает влияния на ход Процесса.

Группа предназначена для группировки графических элементов, принадлежащих одной и той же категории. (Рис. 8.5*в*). Такого рода группировка используется для составления документации или при проведении анализа.

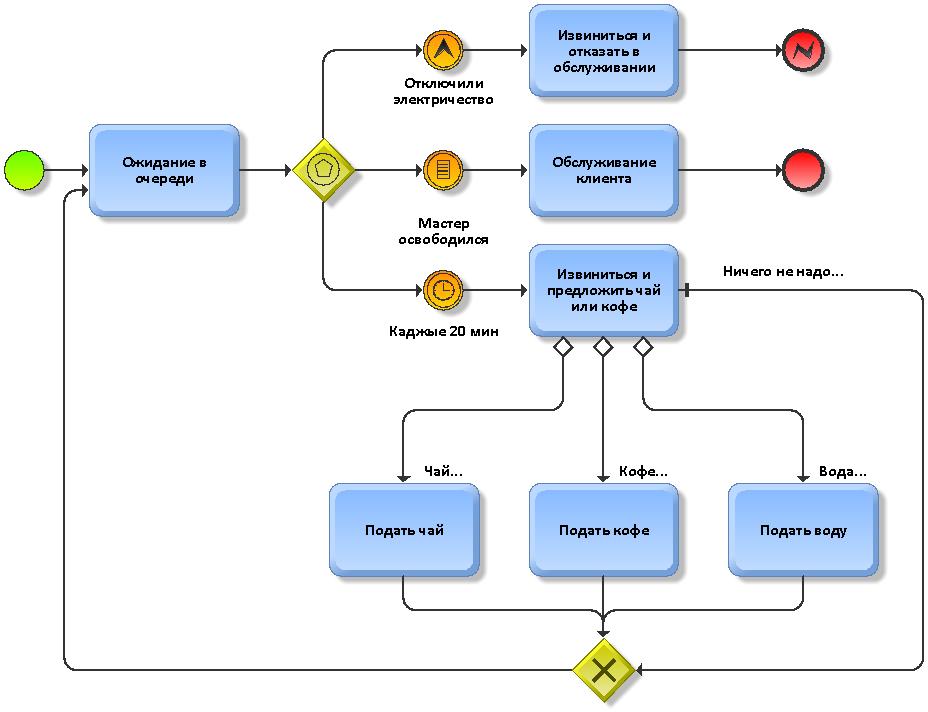
|  |  |  |
| --- | --- | --- |
| Message.jpeg | TextAnnotation.jpeg | Group.jpeg |
| *а* | *б* | *в* |

*Рис. 8.5*. Символы артефактов: *а* - символ сообщения; *б* - символ аннотации; *в* - символ группировки

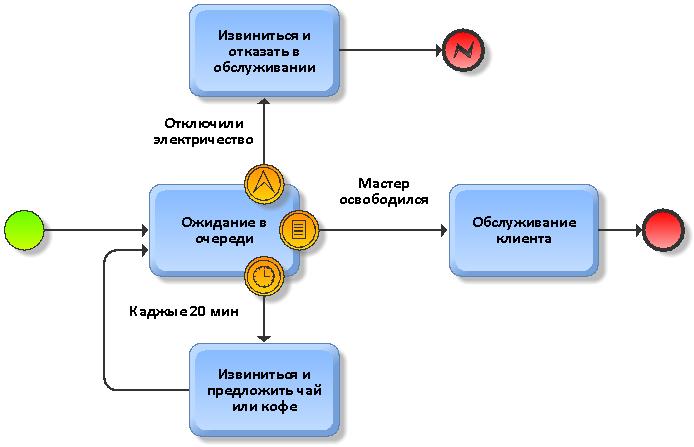
***2. Примеры диаграмм BPMN***



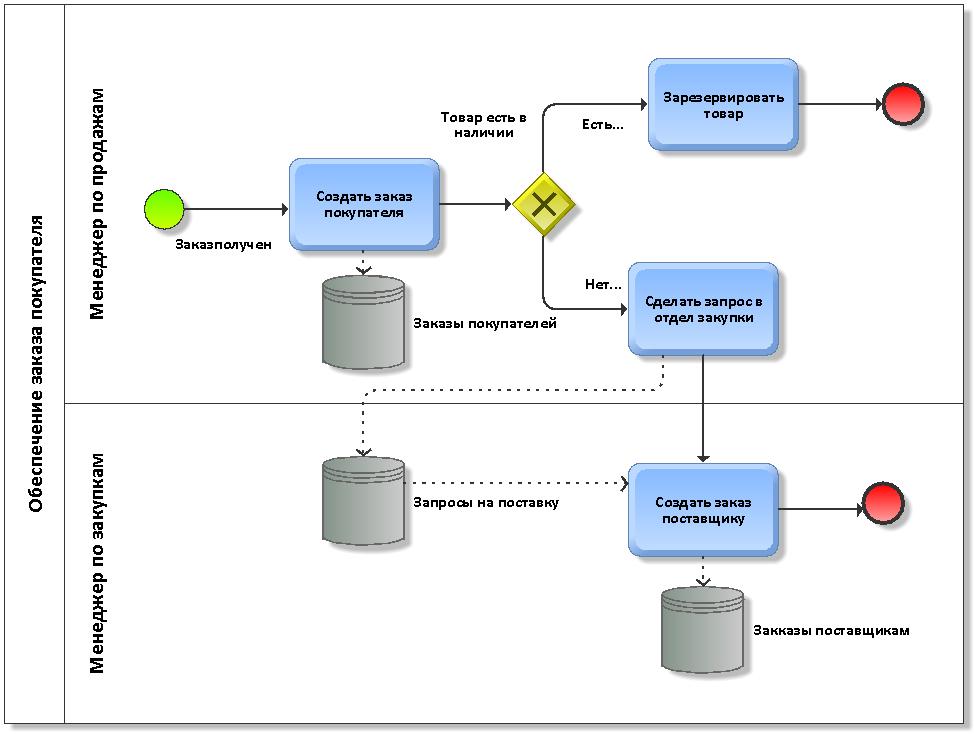
*Рис. 8.6.* Пример использования шлюзов



*Рис. 8.7*. Пример использования специфических потоков управления



*Рис. 8.8.* Пример использования граничных событий



*Рис. 8.9.* Пример процесса обработки заказов в интернет-магазине

### ***2. Рекомендации по построению диаграмм BPMN***

1. Временная последовательность выполнения процессов задается расположением процессов на диаграмме слева направо (сверху вниз на вертикальной диаграмме процесса BPMN).

2. Процесс моделирования подчиняется классическим принципам моделирования: декомпозиции и иерархического упорядочивания.

3. Несмотря на то, что события – необязательные элементы на диаграммах, рекомендуется отображать начальные и конечные события. У одного процесса (пула, дорожки, развернутого подпроцесса) должно быть только одно начальное событие, но может быть несколько конечных событий.

4. На диаграмме не должны присутствовать элементы без единой связи.

5. Каждый шлюз слияния должен обладать минимум двумя входящими связями, шлюз ветвления – минимум двумя исходящими.

6. Количество пересечений линий следует минимизировать. При этом считается, что пересекающиеся линии не имеют логической связи друг с другом. Другими словами, потоки в местах пересечений не меняют своего направления.

7. Шлюз, разветвляющий ветки, и шлюз, объединяющий эти ветки, должны совпадать. Допускается также ситуация, когда шлюз ветвления «И», шлюз объединения – «ИЛИ».

**Содержание отчета**

Диаграммы BPMN и комментарии к ним.

**Контрольные вопросы**

1. В каких случаях на диаграмме используется пул?

2. Какие символы используются для задания альтернативных процессов

3. Какие символы необходимо использовать в процессе декомпозиции процессов?

4. Чем отличаются типы контроля развития процессов эксклюзивных и не эксклюзивных шлюзов?

5. Какие символы используются для представления сопровождения выполнения процессов?

**ПРИЛОЖЕНИЕ 1**

**ИНФОРМАЦИОННАЯ ОЦЕНКА UML ДИАГРАММ**

Количественная оценка информационной ценности UML диаграммы определяется следующими формулами.

**S** – оценка диаграммы;

– оценка элемента диаграммы;

– оценка связи диаграммы;

**Obj** – количество объектов диаграммы;

– число типов объектов;

- число типов связей.

Для диаграмм, содержащих большое число связей одного типа, тип связи не учитывается:

Оценки элементов и связей диаграммы определяется их типом (Табл.1).

При оценке классов, содержащих атрибуты и операции, к оценке класса прибавляется

– оценка операций и атрибутов в классе;

Op – число операций класса;

Atr – число атрибутов класса.

**Оценки элементов и связей UML диаграмм**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Диаграмма | | Прецедентов | Классов | Состояний | Деятельности | Последовательности | Объектов | Компонентов | Размещения | Пакетов | Кооперации |
| **Элементы** | Прецедент, актер | 2 |  |  |  |  |  |  |  |  |  |
| Класс, объект |  | 5 |  |  | 5 |  |  |  |  | 5 |
| Интерфейс |  | 4 |  |  |  |  | 4 |  |  |  |
| Пакет |  | 4 |  |  |  |  |  |  | 4 |  |
| Состояние |  |  | 4 | 4 |  |  |  |  |  |  |
| Компонент |  |  |  |  |  |  | 4 |  |  |  |
| Узел |  |  |  |  |  |  |  | 3 |  |  |
| Процессор |  |  |  |  |  |  |  |  |  |  |
| Взаимодействие |  |  |  |  |  |  |  | 6 |  |  |
| Примечание | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Связи** | Взаимодействие | 1 |  |  |  |  |  |  |  |  |  |
| Расшитение | 1 |  |  |  |  |  |  |  |  |  |
| Обобщение | 3 | 3 |  |  |  |  |  |  |  |  |
| Включение | 1 |  |  |  |  |  |  |  |  |  |
| Зависимость |  | 2 |  |  |  |  |  |  |  |  |
| Ассоциация |  | 1 |  |  |  | 1 |  | 1 | 1 | 1 |
| Агрегация |  | 2 |  |  |  |  |  |  |  |  |
| Композиция |  | 3 |  |  |  |  |  |  |  |  |
| Обобщение |  | 3 |  |  |  |  |  |  |  |  |
| Переход |  |  | 1 | 1 |  |  |  |  |  |  |
| Сообщение |  |  |  |  | 1 |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| **Рекомендуемый**  **диапазон оценок** | | 2.5-3 | 5-5.5 | 2.5-3 | 2.5-3 | 3.5-3 |  | 3.5-4 | 2-2.5 | 3.5-4 | 3.5-4 |

**ПРИЛОЖЕНИЕ 2**

**ВАРИАНТЫ ЗАДАНИЙ**

1. Информационная система отдела сбыта продукции мясокомбината

2. Информационная система «Управление продажами недвижимости»

3. Информационная система «Детский дом»

4. Информационная система театра

5. Информационная система «Санаторий»

6. Информационная система «Домашняя бухгалтерия»

7. Информационная система «Ремонты оборудования»

8. Информационная система «Гостиница домашних животных»

9. Информационная система «Прокат автомобилей»

10. Информационная система «Паспорт здоровья сотрудника»

11. Информационная система «Банковская карта»

12. Информационная система «Ресторан»

13. Информационная система «Лесозавод»

14. Система электронного документооборота предприятия

15. Информационная система «Учет зданий и сооружений»

16. Информационная система «Распределение выпускников»

17. Информационная система «Организация спортивного мероприятия»

18. Информационная система «Хлебозавод»

19. Информационная система аэропорта

20. Информационная система кирпичного завода

21. Информационная система фирмы по продаже запчастей

22. Информационная система «Ветераны предприятия»

23. Информационная система «Банковские вклады»

24. Информационная система «Диетическое питание»

25. Информационная система «SPA-салон»

26. Информационная система «Автосервис»

27. Информационная система водоканала

28. Информационная система «Фотоуслуги»

29. Информационная система «Дошкольное питание»

30. Информационная система «Школьный буфет»

31. Информационная система продажи подержанных автомобилей

32. Энциклопедия футбольной статистики (игры)

33. Информационная система «Такси»

34. Информационная система «Расчет квартплаты»

35. Информационная система «Адвокатская контора»

36. Информационная система «Больничная аптека»

37. Информационная система «Железнодорожный переезд»

38. Информационная система «Аренда и управление недвижимостью»

39. Информационная система «Дошкольное учреждение»

40. Информационная система отдела доставки почтового отделения

41. Городская справочная информационная система

42. Информационная система «Ателье мод»

43. Информационная система «Комбинат ЖБИ»

44. Информационная система «Фитнес-клуб»

45. Информационная система столовой

46. Информационная система «Музейные фонды»

47. Информационная система модельного агентства

48. Информационная система приемной комиссии университета

49. Информационная система «Управление карьером»

50. Информационная система «Управление строительным производством»

51. Информационная система «Паспорт здоровья ребенка»

52. Информационная система «Издательство»

53. Система продажи товаров через Интернет-магазин

54. Информационная система «Клиническая лаборатория»

55. Информационная система «Мясокомбинат»

56. Информационная система гаража

57. Информационная система колледжа

58. Информационная система товарищества собственников жилья

59. Информационная система «Спецодежда»

60. Информационная система «Салон красоты»

61. Информационная система «Отель»

62. Информационная система «Управление переработкой отходов и вторсырья»

63. Информационная система «Предприятие ЖКХ»

64. Информационная система «Рынок труда»

65. Информационная система «Школьная столовая»

66. Информационная система зоопарка

67. Информационная система «Энергосбыт»

68. Информационная система регистрации пассажиров аэропорта

69. Информационная система «Молокозавод»

70. Информационная система «Музей»

71. Информационная система «Библиотека»

72. Информационная система транспортного предприятия

73. Информационная система «Птицеферма»

74. Информационная система страховой компании

75. Информационная система ювелирной мастерской

76. Информационная система платной автостоянки

77. Информационная система абонентов сотовой связи

78. Информационная система «Автопарк»

79. Информационная система «Скорая помощь»

80. Информационная система химчистки

81. Информационная система ломбарда

82. Автоматизированная система «Супермаркет»

83. Энциклопедия футбольной статистики (игроки)

84. Социальная сеть

85. Информационная система учета реализации готовой продукции

86. Информационная система парикмахерской

87. Информационная система туристической фирмы

88. Информационная система «Грузовые перевозки»

89. Информационная система платной поликлиники

90. Информационная система дилерской сети магазина компьютерной техники

91. Информационная система рекламной фирмы

92. Информационная система магазина «Оптика»

93. Информационная система «Туристический клуб»

94. Информационная система администратора футбольной команды

95. Информационная система «Новостная лента предприятий города»

96. Информационная система морского пароходства

97. Информационная система питомника

98. Информационная система автошколы

99. Информационная система «Студенческое общежитие»

100. Информационная система «Цикловая комиссия колледжа (преподаватели и предметы)»

101. Информационная система клуба собаководов

СПИСОК ЛИТЕРАТУРЫ

1. Леоненков А.В. Самоучитель UML. 2-е изд. СПб.: БХВ-Петербург, 2004.
2. Леоненков А.В. UML 2. СПб.: БХВ-Петербург, 2007.
3. Фаулер м. UML. Основы. 3-е изд. /пер. с англ. СПб.: Символ-Плюс, 2004.
4. Рамбо Дж., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка. 2-е изд. СПб.: Питер, 2007.
5. Инструментарий ARIS. Методы. Версия 4.1. URL: <http://businessanalysis.osngrad.info/images/BA_images/aris_methods.pdf>
6. Нотация EPC. URL: http://www.businessstudio.ru/[wiki/docs/v4/ doku.php/ru/csdesign/bpmodeling/epc\_notation](http://www.businessstudio.ru/wiki/docs/v4/doku.php/ru/csdesign/bpmodeling/epc_notation)
7. BPMN 2.0. Из чего состоит модель бизнес процесса. URL: <http://rzbpm.ru/knowledge/bpmn-2-0-iz-chego-sostoit-model-biznes-processa.html>
8. 8. Методология BPMN. URL: http://sites.google.com/site/anisimovkvh/ learning/pris/lecture/tema8/tema8\_4
9. Количественная оценка диаграмм UML. URL. <https://studopedia.ru/9_23685_primer-vipolneniya-raboti.html>

**Содержание**

Лабораторная работа №1. 1

Лабораторная работа №2. 9

Лабораторная работа №3. 25

Лабораторная работа №4. 32

Лабораторная работа №5. 39

Лабораторная работа №6. 48

Лабораторная работа №7. 54

Лабораторная работа №8. 66

Приложение 1 81

Приложение 2 ………………………………………………………………..88

Список литературы………………..…………………… ………………………89

Учебно-методическое пособие

Егоров Сергей Сергеевич,

Соничев Александр Викторович.

**Разработка программного обеспечения**

**cредствами UML и Aris**

Издание публикуется в авторской редакции

СПбГЭТУ «ЛЭТИ»

197376, Санкт*-*Петербург, ул. Проф. Попова, 5

1. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. [↑](#footnote-ref-1)
2. Применительно к конкретным языкам программирования могут быть определены дополнительные кванторы видимости. В этом случае подобные дополнения являются расширением базовой нотации и требуют соответствующих пояснений в форме текста на естественном языке или в виде строки-свойства. [↑](#footnote-ref-2)
3. В отличие от отношения обобщения (см. ниже) части системы не наследуют ее свойства и поведение, поскольку являются вполне самостоятельными сущностями, и обладают своими собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого. [↑](#footnote-ref-3)
4. В некоторых нотациях UML предусмотрено обозначение интерфейсов на диаграммах прецедентов. Интерфейс изображается в виде маленького круга, рядом с которым записывается его имя. [↑](#footnote-ref-4)
5. Системы, которые реагируют на внешние действия от других систем или от пользователей, принято называть реактивными. Если внешние действия инициируются в произвольные случайные моменты времени, то поведении модели называется асинхронным. [↑](#footnote-ref-5)
6. Эти значения не могут быть использованы в качестве имен событий [↑](#footnote-ref-6)
7. Иногда после выражения действия может быть записано сообщение в формате:

   *<имя объекта приемника сообщения> '.' <имя посылаемого сообщения> '('<параметр>':'<тип>,')'.* [↑](#footnote-ref-7)
8. Здесь не рассматривались такие конструкции как составные состояния и подсостояния, последовательные и параллельные подсостояния, исторические, синхронизирующие и сложные состояния, а так же некоторые дополнительные конструкции, такие как точки динамического выбора (dynamic choice points) или точки соединения (junction points). Эти элементы, хотя и позволяют моделировать более сложные аспекты динамического управления поведением объекта, но не являются базовыми и по этой причине не включены в используемую тут нотацию. [↑](#footnote-ref-8)
9. Здесь не рассматривались дорожки (swim lanes), входящие в нотацию UML, но отсутствующие в Violet UML Edit. В диаграммах деятельности им обычно отводится вспомогательная роль. Для отображения соответствующего аспекта функционирования разрабатываемых систем могут быть использованы BPMN-диаграммы Aris. [↑](#footnote-ref-9)
10. В нотации Violet Uml Editor (в отличие от UML Rational Rose) на диаграмму не может быть помещен актер (внешний пользователь). [↑](#footnote-ref-10)
11. Имя класса или имя объекта можно опустить,если не указано имя объекта, то двоеточие перед названием класса обязательно! [↑](#footnote-ref-11)
12. В нотации Violet UML Editor (в отличие от UML Rational Rose) не допускается ветвление потока управления (когда из одной точки фокуса управления объекта могут выходить две и более стрелок передачи сообщения). Кроме того отсутствует средства задания циклов – в UML Rational Rose для этого используются так называемые **фреймы взаимодействий** (interaction frames). [↑](#footnote-ref-12)
13. В нотации Violet UML Editor (в отличие от UML Rational Rose) не предусмотрен специальный символ для обозначения момента уничтожения объекта [↑](#footnote-ref-13)
14. Несмотря на то, что *В нотации Violet Uml Editor в качестве имени объекта может быть использован произвольный текст, следует придерживаться представленного здесь правила.* [↑](#footnote-ref-14)
15. Кроме представленных двух вариантов меток начала стрелок в ARIS Express можно использовать 48 вариантов начал и 48 вариантов концов как для сплошных, так и для пунктирных и состоящих из точек стрелок. [↑](#footnote-ref-15)