Министерство науки и образования РФ
Государственное бюджетное образовательное учреждения
высшего профессионального образования
Московский государственный технический университет
имени Н.Э.Баумана

Кафедра Системы обработки информации и управления

А.В.Балдин

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
к выполнению лабораторных работ
по дисциплине
Объектно ориентированное проектирование автоматизированных систем обработки информации и управления
Специальность: 09.04.01

ДИАГРАММЫ UML2

Москва 2019

СОДЕРЖАНИЕ

1.ВВЕДЕНИЕ

Лабораторные работы по дисциплине «Объектно-ориентированное проектирование автоматизированных систем обработки информации и управления» (специальность 09.04.01) имеют своей целью познакомить студентов со средствами описания программных (информационных) проектов.

На лабораторных работах студент должен самостоятельно освоить программные средства, позволяющее формировать описание проекта на языке UML2 в зависимости от того, какую операционную систему он использует в своей работе.

В процессе выполнения лабораторных работ студент должен получить навыки построения диаграмм на языке UML2 для формирования требований к проектируемой информационной системе, описания статических и динамических аспектов ее построения.

В результате выполнения всех лабораторных работ студент должен освоить технологию проектирования программных (информационных) систем на основе объектно ориентированного подхода и получить навыки формирования проектной документации на основе унифицированного языка моделирования UML2.

2.ВВЕДЕНИЕ В ЯЗЫК UML2

Унифицированный язык моделирования (Unified Modeling Language, UML2) – это универсальный язык визуального моделирования систем. Хотя чаще всего UML2 ассоциируется с моделированием объектно ориентированных программных систем, он имеет намного более широкое применение благодаря свойственной ему расширяемости.

Важно понимать, что UML2 не предлагает нам какой-либо методологии моделирования. Конечно, некоторые методические аспекты подразумеваются элементами, составляющими модель UML2, но сам UML2 предоставляет собой лишь визуальный синтаксис, который можно использовать для создания моделей.

UML2 это не методология, это унифицированный язык визуального моделирования.

2.1.Структура UML2

Понимание работы UML2 как визуального языка, начинается с рассмотрения его структуры. Она показана на рис. 2.1. Эта структура включает:

- строительные блоки – основные элементы, отношения и диаграммы UML2-модели;
- общие механизмы – общие UML2-пути достижения определённых целей;
- архитектура – UML2-представление архитектуры системы.

Понимание структуры UML2 дает нам представление о структуре всего изложенного в книге материала. Наличие структуры также указывает на то, что сам UML2 – это спроектированная система с собственной архитектурой.

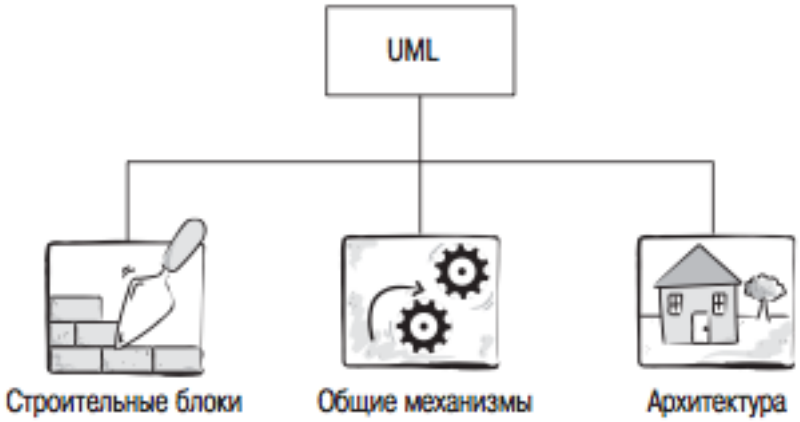


Рис. 2.1. Структура UML2

2.2. Строительные блоки UML2

UML2 состоит всего из трёх строительных блоков (рис. 2.2):

1. Сущности – это сами элементы модели.
2. Отношения связывают сущности. Отношения определяют, как семантически связаны две или более сущностей.
3. Диаграммы – это *представления* моделей UML2. Они показывают наборы сущностей, которые «рассказывают» о программной системе и являются нашим способом визуализации того, *что* будет делать система (аналитические диаграммы) или *как* она будет делать это (проектные диаграммы).

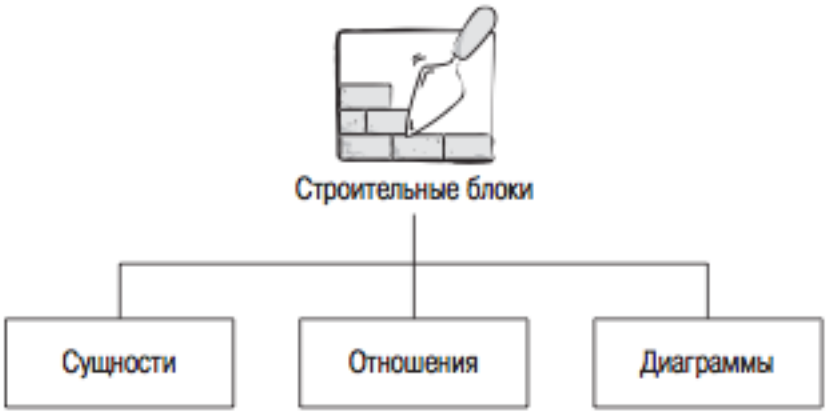


Рис. 2.2. Строительные блоки UML2

2.2.1.Сущности

Все UML2-сущности можно разделить на:

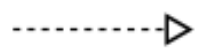
- структурные сущности – существительные UML2-модели, такие как класс, интерфейс, кооперация, прецедент, активный класс, компонент, узел;
- поведенческие сущности – глаголы UML2-модели, такие как взаимодействия, деятельности, автоматы;
- группирующая сущность – пакет, используемый для группировки семантически связанных элементов модели в образующие единое целое модули;
- аннотационная сущность – примечание, которое может быть добавлено к модели для записи специальной информации, очень похожее на стикер.

2.2.2. Отношения

Отношения позволяют показать взаимодействие в пределах модели двух или более сущностей. Для понимания роли, которую отношения играют в моделях UML2, достаточно представить отношения между членами отдельной семьи. Отношения в модели UML2 позволяют зафиксировать значимые (семантические) связи между сущностями. Например, в табл. 2.1 представлены UML2-отношения, применяемые к структурным и группирующим сущностям модели. Правильное понимание семантики различных типов отношений является очень важной частью моделирования UML2.

Таблица 2.1

Отношения в языке UML2

Тип отношения	UML-синтаксис		Краткая семантика
	источник	цель	
Зависимость			Исходный элемент зависит от целевого элемента и изменение последнего может повлиять на первый.
Ассоциация			Описание набора связей между объектами.
Агрегация			Целевой элемент является частью исходного элемента.
Композиция			Строгая (более ограниченная) форма агрегирования.
Включение			Исходный элемент содержит целевой элемент.
Обобщение			Исходный элемент является специализацией более обобщенного целевого элемента и может замещать его.
Реализация			Исходный элемент гарантированно выполняет контракт, определенный целевым элементом.

2.2.3.Диаграммы

Во всех инструментальных средствах UML2 моделирования новые сущности или новые отношения при создании добавляются в модель. Модель – это хранилище всех сущностей и отношений, созданных для описания требуемого поведения проектируемой программной системы.

Диаграммы – это своего рода *картины,* или *представления* модели. Диаграмма это *не* модель! На самом деле, различие между диаграммой и моделью является очень важным для понимания, поскольку сущность или отношение могут быть удалены с диаграммы, или даже со всех диаграмм, но по-прежнему они продолжают существовать в модели. Они будут оставаться в модели до тех пор, пока не будут явно удалены из нее. Общая ошибка новичков в UML2 моделировании состоит в том, что они удаляют сущности с диаграмм, не удаляя их из модели.

Существует тринадцать различных типов UML2 диаграмм, все они приведены на рис. 2.3. На рисунке каждый прямоугольник представляет определенный тип диаграммы, при этом курсивом выделяются абстрактные категории типов диаграмм. Например, существует шесть разных типов Диаграмм структуры. Обычный текст указывает на конкретную диаграмму, которую можно реально создать. Заштрихованные блоки обозначают типы диаграмм, впервые появившиеся в UML2.

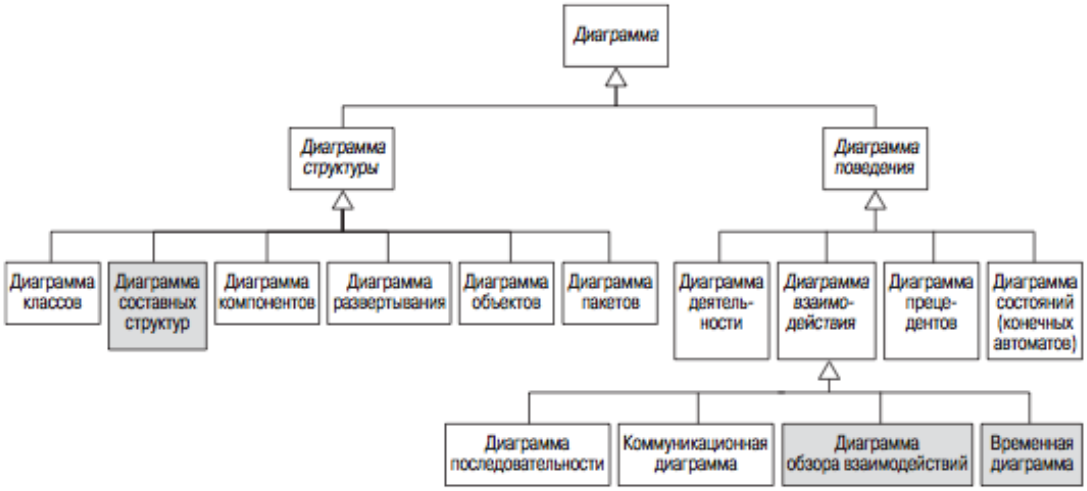


Рис. 2.3. Типы диаграмм UML2

Эти диаграммы можно разделить на те, которые моделируют статическую структуру системы (статическую модель), и те, которые моделируют динамическую структуру системы (динамическую модель).

Статическая модель фиксирует сущности и структурные отношения между ними; динамическая модель отображает, как сущности взаимодействуют для генерирования требуемого поведения программной системы.

Определенного порядка создания UML2 диаграмм не существует, хотя обычно начинают с диаграммы прецедентов для определения предметной области системы. Как правило, работа идет одновременно над несколькими диаграммами, каждая из которых уточняется по мере выявления более подробной информации о разрабатываемой программной системе. Таким образом, диаграммы являются как представлением модели, так и основным механизмом введения информации в модель.

В UML2 представлен новый синтаксис диаграмм, изображенный на рис. 2.4. У каждой диаграммы может быть рамка, область заголовка и область содержимого. Область заголовка – это неправильный

пятиугольник, содержащий тип (не обязательно), имя и параметры (не обязательно) диаграммы.

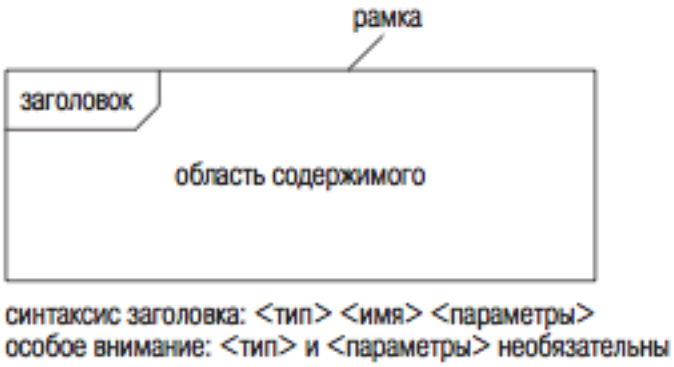


Рис. 2.4. Синтаксис UML2 диаграмм

2.3.Общие механизмы UML2

В UML2 существует четыре общих механизма, последовательно применяемых ко всему языку моделирования. Они описывают четыре стратегии подхода к моделированию объектов, которые в разных контекстах многократно применяются в UML2. (рис. 2.5).

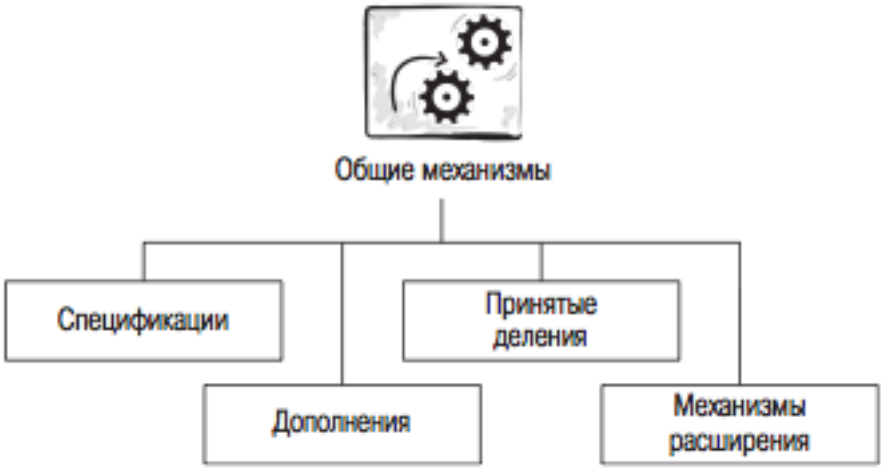


Рис. 2.5. Общие механизмы UML2

2.3.1. Спецификации

Модели UML2 имеют, по крайней мере, два измерения: графическое, позволяющее визуализировать модель с помощью диаграмм и пиктограмм, и текстовое, состоящее из спецификаций

различных элементов модели. Спецификации – это текстовые описания семантики элемента.

2.3.2.Дополнения

В UML2 каждый элемент модели обозначается простым символом, к которому можно добавлять ряд дополнений, визуализирующих аспекты спецификации элемента. С помощью этого механизма видимая на диаграмме информация может быть представлена в соответствии с конкретными требованиями.

2.3.3.Принятые деления

Принятые деления описывают конкретные способы представления мира. В UML2 существует два принятых деления: классификатор/экземпляр и интерфейс/реализация.

Классификатор и экземпляр

В UML2 предполагается, что может существовать абстрактное понятие типа сущности (например, банковский счет) и отдельные конкретные экземпляры этой абстракции (такие как «мой банковский счет» или «ваш банковский счет»). Абстрактное понятие типа сущности – это классификатор, а отдельные конкретные сущности – экземпляры.

Интерфейс и реализация

Основная идея этих понятий в том, чтобы отделить то, *что* выполняет действие (интерфейс), от того, *как* это делается (реализации).

2.4.Механизмы расширения

Разработчики UML2 понимали, что просто невозможно создать полностью универсальный язык моделирования, который удовлетворял бы всем современным требованиям и тем, что могут появиться в ближайшем будущем. Поэтому UML2 включает три простых механизма расширения, приведенные в табл. 2.2.

Таблица 2.2

Механизмы расширения

	Механизмы расширения UML
Ограничения	Расширяют семантику элемента, обеспечивая возможность добавлять новые правила.
Стереотипы	Обеспечивают возможность определять новые элементы модели UML на основании существующих: мы определяем семантику стереотипа самостоятельно. Стереотипы добавляют новый элемент в метамодель UML.
Помеченные значения	Предоставляют способ расширения спецификации элемента, обеспечивая возможность добавлять в него новую специальную информацию.

Ограничения

Ограничение – это строка текста, заключённая в фигурные скобки { }, определяющая некоторое условие или правило для элемента модели, которое *должно* оставаться истинным. Иначе говоря, оно некоторым образом ограничивает какие-либо свойства элемента.

Стереотипы

Стереотип представляет разновидность существующего элемента модели, имеющего ту же форму (например, атрибуты и отношения), но другое назначение.

Стереотипы позволяют создавать новые элементы модели на основании *существующих.* Для этого к имени нового элемента добавляется имя стереотипа угловых скобках «...». Число стереотипов каждого элемента модели может изменяться от нуля до некоторого значения.

Помеченные значения

В UML2 свойство – это любое значение, прикреплённое к элементу модели. Большинство элементов имеют большое число предопределённых свойств. Некоторые из них могут отображаться на диаграммах, другие являются частью семантического заднего плана модели.

3.ЛАБОРАТОРНАЯ РАБОТА № 1. МОДЕЛИРОВАНИЕ ТРЕБОВАНИЙ К ПРОГРАММНОЙ СИСТЕМЕ

Цель работы: Целью работы является знакомство с программной средой формирования диаграмм UML2 и определение требований к проектированию программной системы посредством построения диаграммы прецедентов

3.1.Контекст системы (границы системы)

Первое, что необходимо сделать при построении системы, – обозначить ее границы. Иначе говоря, надо определить, что является *частью* системы (находится внутри границ системы) и что находится *вне* системы (вне ее границ). Это кажется очевидным, но встречается немало проектов, в которых из-за неясности границы системы возникают серьёзные проблемы. Точное определение границ системы обычно играет важную роль в выявлении функциональных (а иногда и нефункциональных) требований. В UML2 границу системы называют *контекстом системы* (subject).

Контекст системы определяется тем, кто или что использует систему (т. е. актёрами), и тем, какие конкретные преимущества система предлагает этим актёрам (т. е. прецедентами).

Актёр (действующее лицо) определяет роль, которую выполняет некоторая внешняя сущность при *непосредственном* взаимодействии с данной системой. Он может представлять роль пользователя или роль, исполняемую другой системой или частью аппаратных средств, которые касаются границ системы.

Актеры – это роли, исполняемые сущностями, непосредственно взаимодействующими с системой.

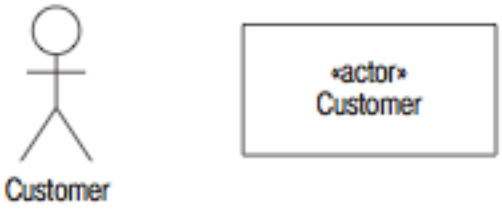


Рис. 3.1. Варианты изображения актера

При моделировании актеров необходимо помнить следующие моменты.

- Актеры всегда являются **внешними** по отношению к системе, следовательно, **находятся вне вашего контроля**.

- Актеры взаимодействуют *непосредственно* с системой – так они помогают в определении контекста системы.
- Актеры представляют роли, исполняемые людьми или сущностями по отношению к системе, а не конкретных людей или сущностей.
- Один человек или сущность может играть по отношению к системе множество ролей одновременно или последовательно во времени. Например, вы составляете и ведёте учебные курсы. С точки зрения системы планирования курсов вы играете две роли: Trainer (инструктор) и CourseAuthor (автор курса).
- У каждого актёра должно быть короткое, осмысленное с прикладной точки зрения имя.
- Каждого актёра должно сопровождать краткое описание (одна или две строчки), объясняющее, что данный актер из себя представляет с прикладной точки зрения.

Прецедент – это что-то, что должна делать система по желанию актёра. Это «вариант использования» системы конкретным актёром:

- прецеденты *всегда* инициируются актёром;
- прецеденты *всегда* описываются с точки зрения актёров.

Рис. 3.2. Пиктограмма прецедента

3.2.Диаграмма прецедентов

На диаграмме прецедентов контекст модели прецедентов изображается в виде блока с именем контекста. Этот блок является контекстом и представляет границу системы, моделируемую прецедентами. Актёры располагаются вне контекста (они внешние по отношению к системе), а прецеденты, составляющие поведение системы, располагаются внутри контекста (они внутренние по отношению к системе). Это проиллюстрировано на рис. 3.3.

Отношение между актёром и прецедентом обозначается сплошной линией. Это символ ассоциации в UML2. Ассоциация между актером и прецедентом показывает, что актёр и прецедент каким-то образом взаимодействуют.

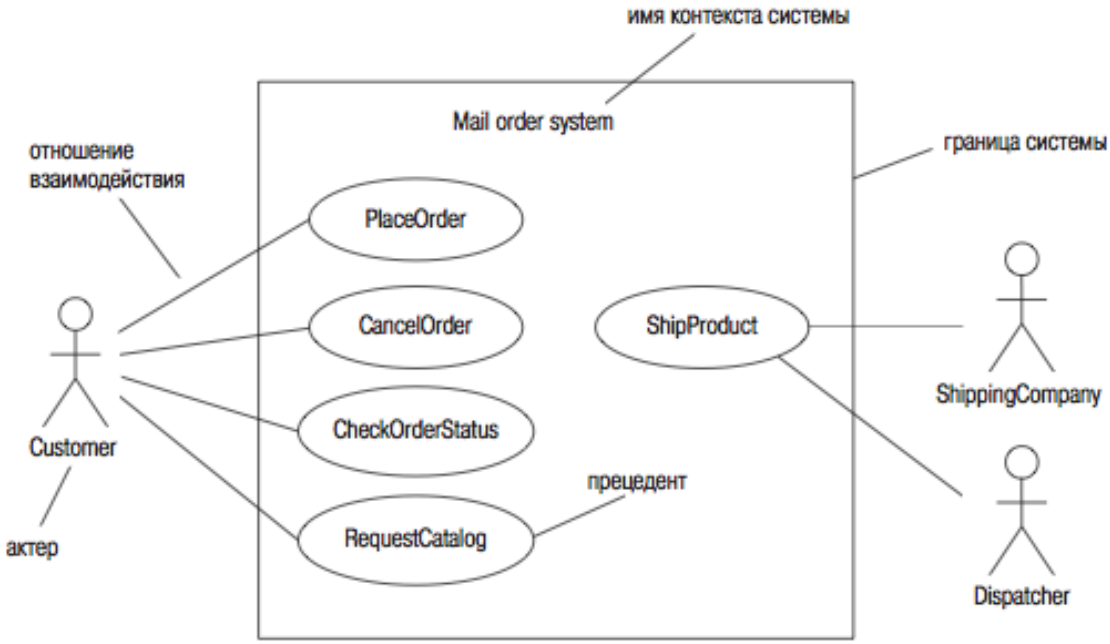


Рис. 3.3. Диаграмма прецедентов

3.3.Задание на лабораторную работу.

- Изучить теоретическую часть лабораторной работы.
- Для выбранной предметной области определить границы информационной (программной) системы и сформулировать требования к ней.
- Оформить требования в виде диаграммы прецедентов.
- Оформить отчёт по лабораторной работе

3.4.Требования к содержанию отчёта

Отчёт по лабораторной работе должен содержать:

- Название работы
- Цель работы
- Краткое описание предметной области, для которой строится программная система
- Перечень требований к проектируемой программной системе
- Диаграмма прецедентов, реализующая представленные требования
- Выводы по работе

4.ЛАБОРАТОРНАЯ РАБОТА № 2. МОДЕЛИРОВАНИЕ СТАТИЧЕСКОЙ СТРУКТУРЫ ПРОГРАММНОЙ СИСТЕМЫ

Цель работы: Целью работы является построение статической структуры программных модулей системы посредством построения диаграммы компонентов и структуры всей системы в виде диаграммы пакетов, а также построение модели развёртывания программной системы на технических средствах в виде диаграммы развёртывания.

4.1.Диаграмма компонентов

4.1.1.Компонент

Компонент – модульная и замещаемая часть системы, инкапсулирующая ее содержимое.

Компонент как чёрный ящик, внешнее поведение которого полностью определяется его предоставляемыми и требуемыми интерфейсами. Поэтому один компонент может быть заменён другим, поддерживающим тот же протокол.

Компоненты могут иметь атрибуты и операции и участвовать в отношениях ассоциации и обобщения. Компоненты – это структурированные классификаторы. У них может быть внутренняя структура, включающая части и соединители.

На диаграмме компонентов могут быть показаны компоненты, зависимости между ними и то, как компонентам назначаются классификаторы. Компонент отображается в виде прямоугольника со стереотипом «component» (компонент) и/или пиктограммой компонента в верхнем правом углу. У компонентов могут быть предоставляемые и требуемые интерфейсы и порты.

Компонент может иметь внутреннюю структуру. Части можно показать вложенными внутрь компонента или находящимися снаружи и соединёнными с ним отношением зависимости. Обе формы синтаксически эквивалентны, хотя первая нотация более наглядна.

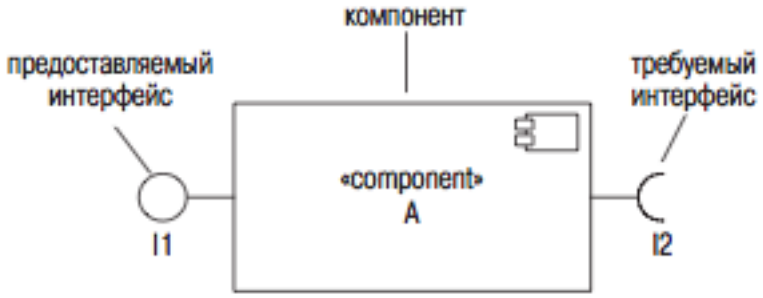


Рис. 4.1. Нотация компонента

Если у компонента есть внутренняя структура, как правило, он будет делегировать обязанности, определенные его интерфейсами, своим внутренним частям. На рис. 4.2 компонент A предоставляет интерфейс I1 и требует интерфейс I2. Он инкапсулирует две части типа b и c. Он делегирует поведение, описанное его предоставляемым и требумым интерфейсами b и c соответственно.

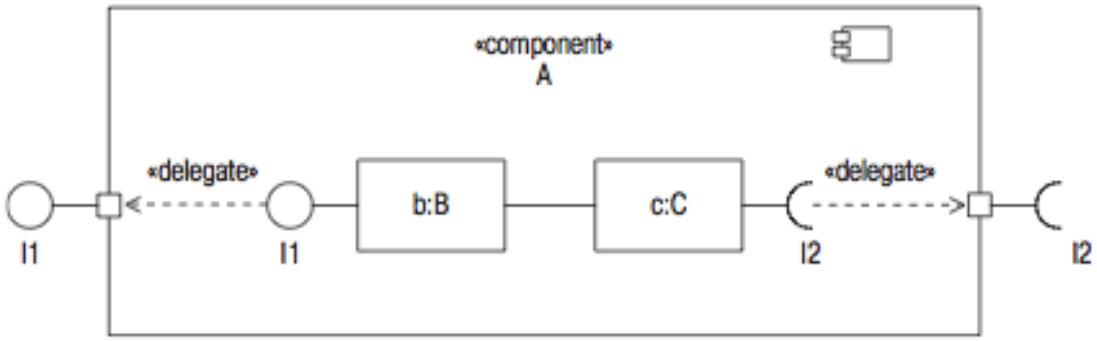


Рис. 4.2. Внутренняя структура компонента

4.1.2.Подсистемы

Подсистема – это компонент, действующий как единица декомпозиции большой системы. Подсистемы изображаются как компонент со стереотипом «subsystem».

Подсистема – это логическая конструкция, используемая для декомпозиции большой системы в управляемые части. Экземпляры самих подсистем *не могут* создаваться во время выполнения, но могут создаваться экземпляры их содержимого.

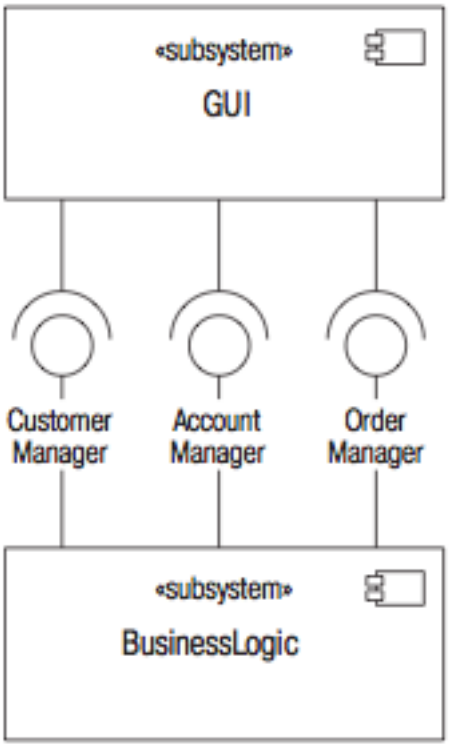


Рис. 4.3. Подсистемы и интерфейсы

4.2.Диаграмма пакетов

4.2.1.Пакеты

Пакет – это группирующая сущность. Это контейнер и владелец элементов модели. У каждого пакета есть своё пространство имён, в рамках которого все имена должны быть уникальными.

По сути, пакет – это универсальный механизм организации элементов модели (включая другие пакеты) и диаграмм в группы. Он может использоваться для следующих целей:

- предоставления инкапсулированного пространства имён, в рамках которого все имена должны быть уникальными;
- группировки семантически взаимосвязанных элементов;
- определения «семантической границы» модели;
- предоставления элементов для параллельной работы и управления конфигурацией.

Пакеты позволяют создавать допускающую навигацию хорошо структурированную модель, обеспечивая возможность группировать сущности, имеющие близкие семантические связи. В модели можно устанавливать семантические границы, в пределах которых разные пакеты описывают разные аспекты функциональности системы.

Пакет определяет так называемое инкапсулированное пространство имён. Это означает, что пакет создаёт границу, в рамках которой имена всех элементов должны быть уникальными. Это также означает, что если элементу из одного пространства имён необходимо обратиться к элементу из другого пространства имён, он должен указать и имя необходимого элемента, *и* путь к этому элементу, чтобы его можно было найти в пространствах имён.

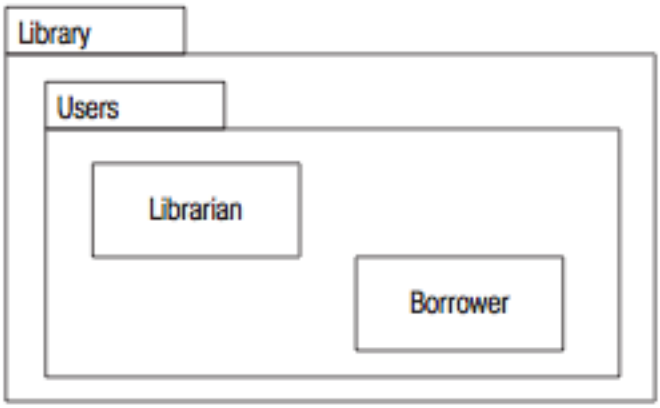


Рис. 4.4. Синтаксис вложения путём встраивания

4.2.2.Вложенные пакеты

Пакеты могут быть вложены в другие пакеты с любой глубиной вложенности. Однако обычно достаточно всего двух или трёх уровней. В противном случае модель может стать трудной для понимания и в ней будет сложно ориентироваться.

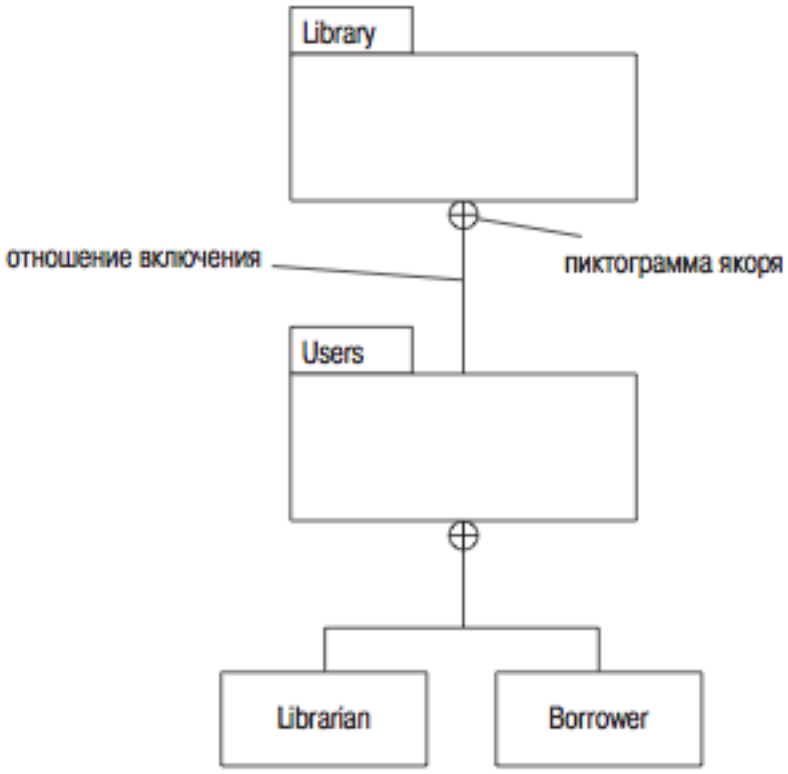


Рис. 4.5. Синтаксис сложного вложения

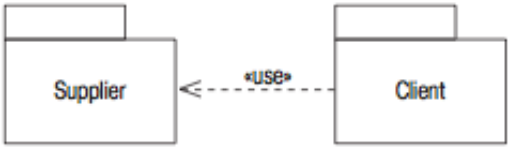
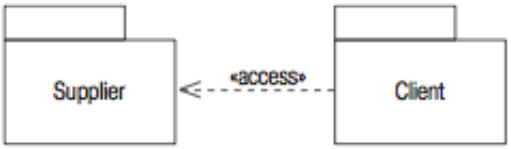
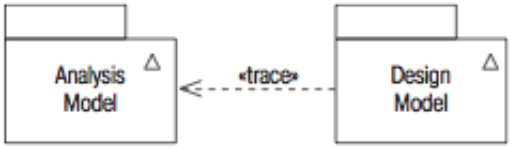
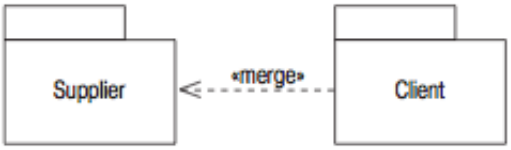
4.2.3.Зависимости пакетов

Между пакетами может быть установлено отношение зависимости.

И как всегда, при создании аналитической модели пакетов необходимо придерживаться простоты. Важнее получить правильный набор пакетов, чем широко применять такие возможности, как обобщение пакетов и стереотипы зависимостей. Все это можно добавить позже и только в том случае, если это сделает модель более понятной.

Таблица 4.1.

Отношения на диаграмме пакетов

Отношение зависимости пакетов	Семантика
	Элемент клиентского пакета некоторым образом использует открытый элемент пакета-поставщика – клиент зависит от поставщика. Если зависимость пакета показана без стереотипа, необходимо предполагать зависимость «use».
	Открытые элементы пространства имен поставщика добавляются как открытые элементы в пространство имен клиента. Элементы клиента могут организовывать доступ ко всем открытым элементам поставщика через неполные имена.
	Открытые элементы пространства имен поставщика добавляются как закрытые элементы в пространство имен клиента Элементы клиента могут организовывать доступ ко всем открытым элементам поставщика с помощью неполных имен.
	«trace», как правило, представляет историческое развитие одного элемента в другую более развитую версию; обычно это отношение между моделями, а не элементами (межмодельное отношение).
	Открытые элементы пакета-поставщика объединяются с элементами клиентского пакета. Эта зависимость используется только при создании метамодели; в обычном ОО анализе и проектировании она не должна встречаться.

Отсутствие вложенных пакетов – один из гарантов простоты модели. Чем глубже что-то помещено в структуру вложенных пакетов, тем более непонятным оно становится.

4.3.Диаграмма развёртывания

Диаграмма развёртывания проецирует программную архитектуру на аппаратную архитектуру.

Диаграмма развёртывания определяет физическое оборудование, на котором будет выполняться программная система, а также описывает, как программное обеспечение развёртывается на это оборудование.

Диаграмма развёртывания проецирует программную архитектуру, созданную при проектировании, на исполняющую ее физическую архитектуру системы. В распределённых системах она моделирует распределение программного обеспечения по физическим узлам.

Существует две формы диаграмм развертывания.

1. Дескрипторная форма (descriptor form) – содержит узлы, отношения между узлами и артефакты. Узел представляет тип оборудования (например, ПК). Аналогично артефакт представляет тип физического программного артефакта, например Java JAR□файл.
2. Экземплярная форма (instance form) – включает экземпляры узлов, отношения между экземплярами узлов и экземпляры артефактов. Экземпляры узлов представляют конкретную, идентифицируемую часть оборудования (например, ПК Джима). Экземпляр артефакта представляет конкретный экземпляр типа программного обеспечения, например, конкретный JAR файл. Если детали конкретных экземпляров неизвестны (или неважны), могут использоваться анонимные экземпляры.

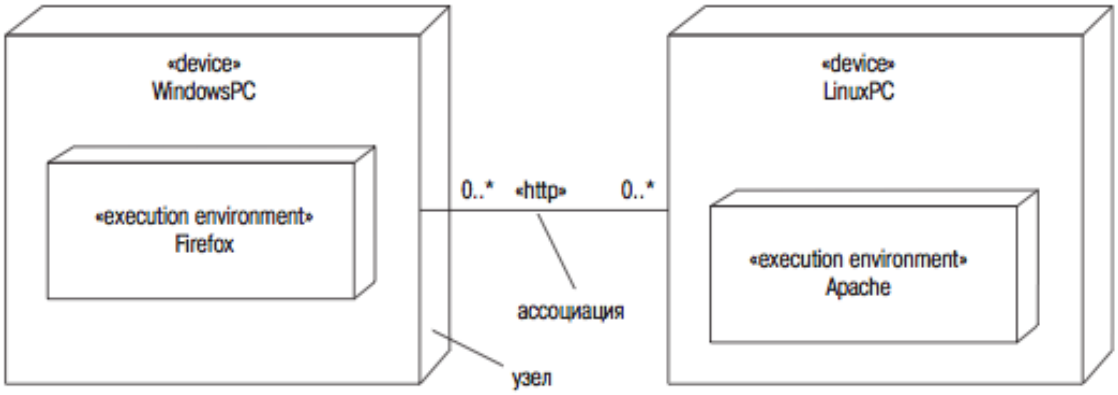


Рис. 4.6. Дескрипторная форма диаграммы развёртывания

Дескрипторная форма диаграмм развёртывания хороша для моделирования типа физической архитектуры, а экземплярная форма –

для моделирования фактического развёртывания этой архитектуры на конкретном сайте.

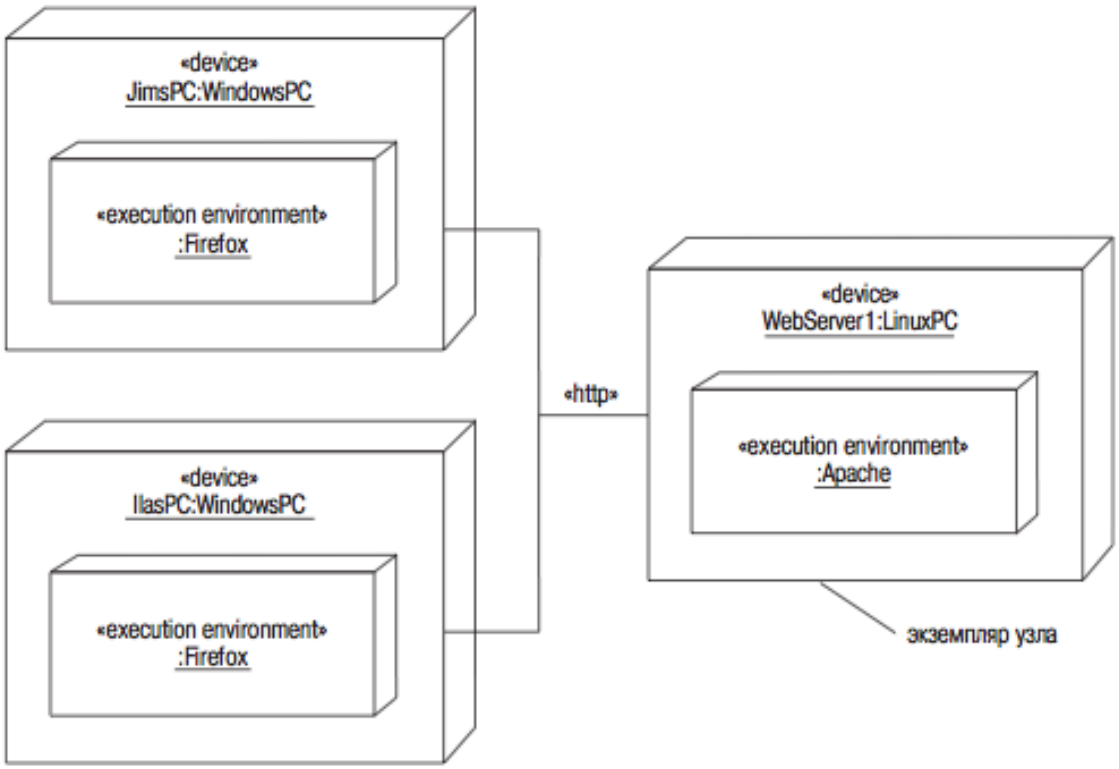


Рис. 4.7. Экземплярная форма диаграммы развёртывания

Для стереотипов можно придумать собственные пиктограммы, напоминающие реальное оборудование, и затем использовать эти символы на диаграмме развёртывания. Такой подход упрощает восприятие диаграммы. Пример полностью визуализированной дескрипторной формы диаграммы развертывания приведён на рис. 4.8.

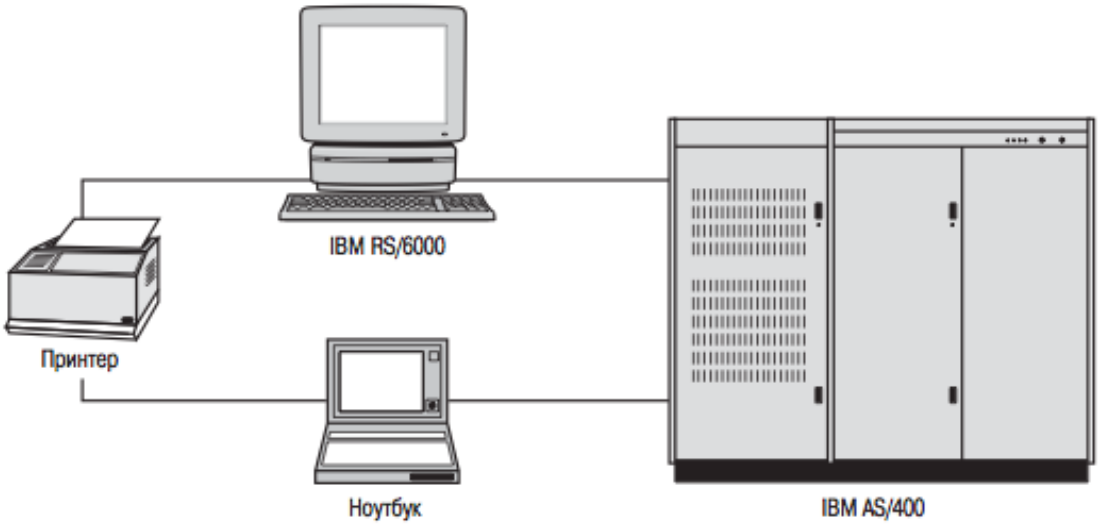


Рис. 4.8. Визуализированная дескрипторная форма диаграммы развёртывания

4.4.Задание на лабораторную работу.

- Изучить теоретическую часть лабораторной работы.
- Для выбранной предметной области определить структуру программной системы в виде диаграммы компонент.
- Оформить архитектуру программной системы в виде диаграммы пакетов.
- Сформировать техническую среду развёртывания программной системы в виде диаграммы развёртывания.
- Оформить отчёт по лабораторной работе

4.5.Требования к содержанию отчёта

Отчёт по лабораторной работе должен содержать:

- Название работы
- Цель работы
- Краткое описание предметной области
- Диаграмма компонентов, отражающая состав программных модулей системы
- Диаграмма пакетов, отражающая структуру программной системы
- Диаграмма развёртывания, определяющая состав технических средств и размещение программных пакетов на них
- Выводы по работе

5.ЛАБОРАТОРНАЯ РАБОТА № 3. МОДЕЛИРОВАНИЕ СТАТИЧЕСКОЙ СТРУКТУРЫ ПРОГРАММНОГО МОДУЛЯ

Цель работы: Целью работы является построение статической структуры программного модуля посредством построения диаграммы классов, формирования описания структуры базы данных и пользовательского интерфейса на основе диаграммы классов

5.1.Диаграмма классов

5.1.1.Нотация классов в UML2

Визуальный синтаксис UML2 для класса очень богат. Чтобы синтаксис был управляемым, в UML2 существует понятие необязательных дополнений. Обязательной частью в визуальном синтаксисе является только ячейка с именем класса. Все остальные ячейки и дополнения необязательны. Для справки на рис. 5.1. показано все.

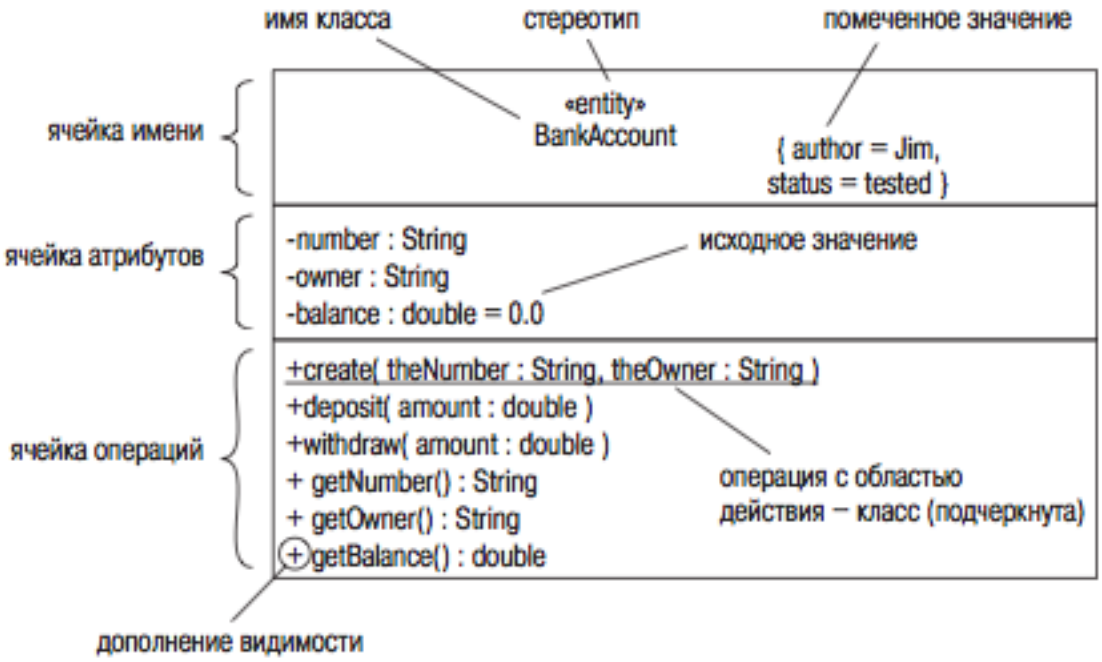


Рис. 5.1. Нотация классов в UML2

В аналитических моделях обычно необходимо показывать только:

- имя класса;
- ключевые атрибуты;
- ключевые операции;

- стереотипы (если они приносят пользу делу).

Обычно *не* показывают следующее:

- помеченные значения;
- параметры операций;
- видимость;
- исходные значения (если только они не значимы для дела).

Используя общую структуру описания классов в языке UML2 возможно построить модели не только классов, но и таблиц баз данных и интерфейсов.

При использовании ячейки имени (обязательной для любой диаграммы классов) и ячейки атрибутов, опуская ячейку операций, получаем описание таблиц базы данных в нотации диаграммы классов.

При использовании ячейки имени и ячейки операций, опуская ячейку атрибутов, получаем описание интерфейсов, или описание структуры диалогового взаимодействия пользователя с системой, в нотации диаграммы классов.

5.1.2.Отношения

Отношения – это семантические (значимые) связи между элементами модели. Отношения – это способ объединения сущностей в UML2.

Обычно одна связь соединяет только два объекта. Однако UML2 допускает соединение нескольких объектов одной связью. Такую связь называют n-арной. Она изображается в виде ромба, от которого отходят линии к каждому из объектов-участников.

Спецификация UML2 допускает три разных способа отображения возможности навигации. Самое распространённое обозначение:

- все кресты опускаются;
- двунаправленные ассоциации изображаются *без* стрелок;
- однонаправленные ассоциации изображаются с одной стрелкой.

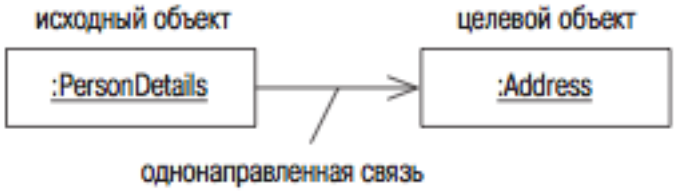


Рис. 5.2. Однонаправленная связь между объектами

Ассоциации – это отношения между классами. Аналогично связям, соединяющим объекты, ассоциации соединяют классы. Самое главное: для того чтобы между двумя объектами была связь, между классами этих объектов *должна* существовать ассоциация. Потому что связь – это экземпляр ассоциации. Так же как объект – экземпляр класса.

Семантика базовой, неуточненной ассоциации чрезвычайно проста: ассоциация между классами указывает на то, что между объектами этих классов могут устанавливаться связи. Существуют другие, более конкретные формы ассоциаций (агрегация и композиция.

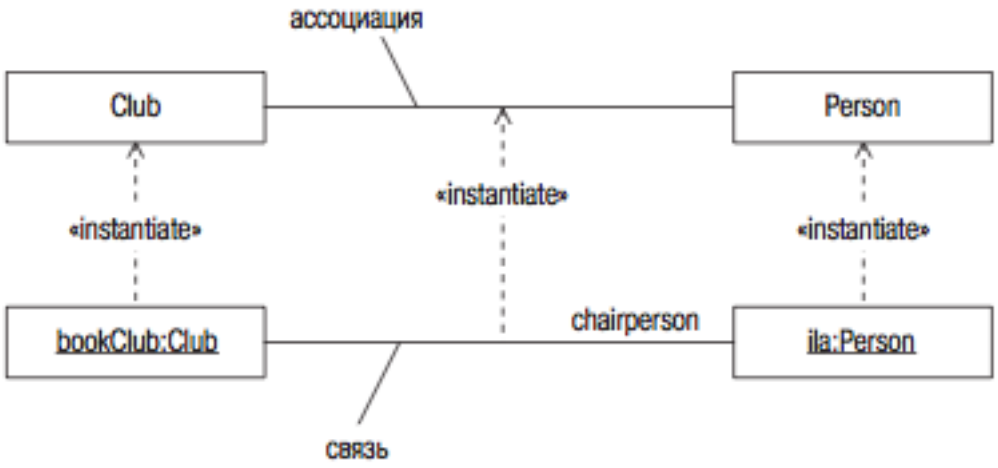


Рис. 5.3. Отношения между классами и объектами и между связями и ассоциациями

Ассоциации могут иметь:

- имя ассоциации
- имена ролей
- кратность
- возможность навигации

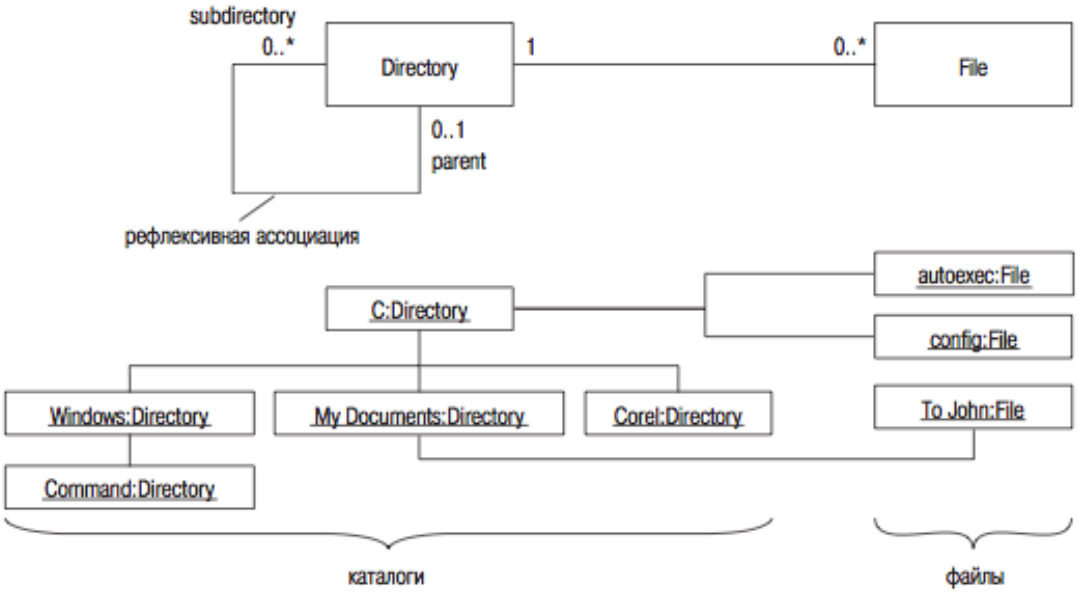


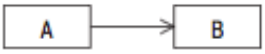
Рис. 5.4. Пример рефлексивной ассоциации: вверху – диаграмма классов, внизу – диаграмма объектов

5.1.3.Возможность навигации

Возможность навигации (navigability) указывает на возможность прохода от объекта исходного класса к одному или более объектам в зависимости от кратности целевого класса. Смысл навигации в том, что «сообщения могут посылаться только в направлении, в котором указывает стрелка».

Таблица 5.1

Стили представления возможности навигации в UML2

Стили представления возможности навигации в UML 2			
Синтаксис UML 2	Стиль 1: Строгое обозначение возможности навигации в UML 2	Стиль 2: Возможность навигации не обозначается	Стиль 3: Общепринятая практика
	Есть возможность навигации от A к B Есть возможность навигации от B к A		
	Есть возможность навигации от A к B Нет возможности навигации от B к A		
	Есть возможность навигации от A к B Возможность навигации от B к A не определена		Есть возможность навигации от A к B Нет возможности навигации от B к A
	Возможность навигации от A к B не определена Возможность навигации от B к A не определена	Возможность навигации от A к B не определена Возможность навигации от B к A не определена	Есть возможность навигации от A к B Есть возможность навигации от B к A
	Нет возможности навигации от A к B Нет возможности навигации от B к A		

5.1.4.Квалифицированные ассоциации

Квалифицированные ассоциации могут использоваться для превращения ассоциации n ко многим в ассоциацию n к одному путём задания одного объекта (или группы объектов) из целевого набора. Это очень полезные элементы модели, поскольку они показывают, как можно вести поиск или осуществлять навигацию к конкретным объектам коллекции.

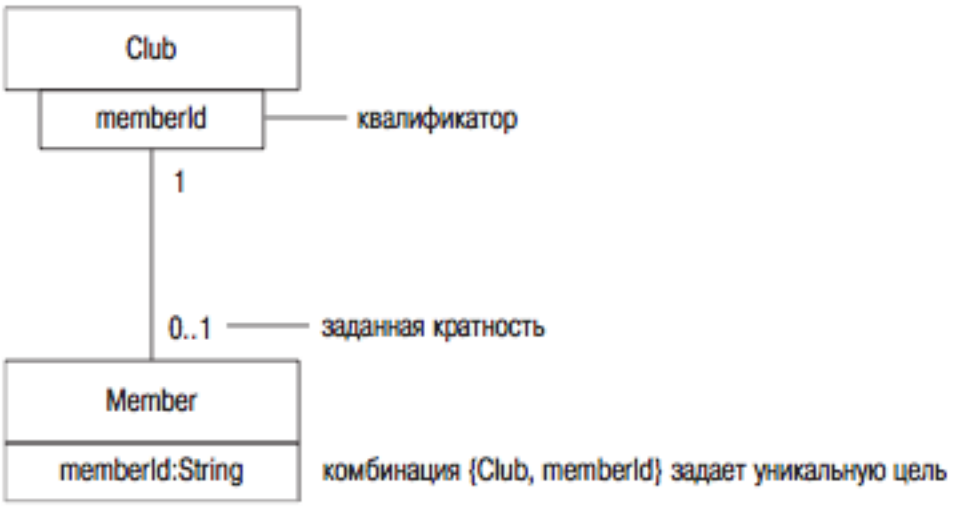


Рис. 5.5. Квалификатор превращает отношение один ко многим в отношение один к одному

5.1.5.Зависимость

Зависимость обозначает отношение между двумя или более элементами модели, при котором изменение одного элемента (поставщика) может повлиять или предоставить информацию, необходимую другому элементу (клиенту). Иначе говоря, клиент некоторым образом зависит от поставщика. Зависимости используются для моделирования отношений между классификаторами, когда один классификатор зависит от другого, но отношение не является ни ассоциацией, ни обобщением.

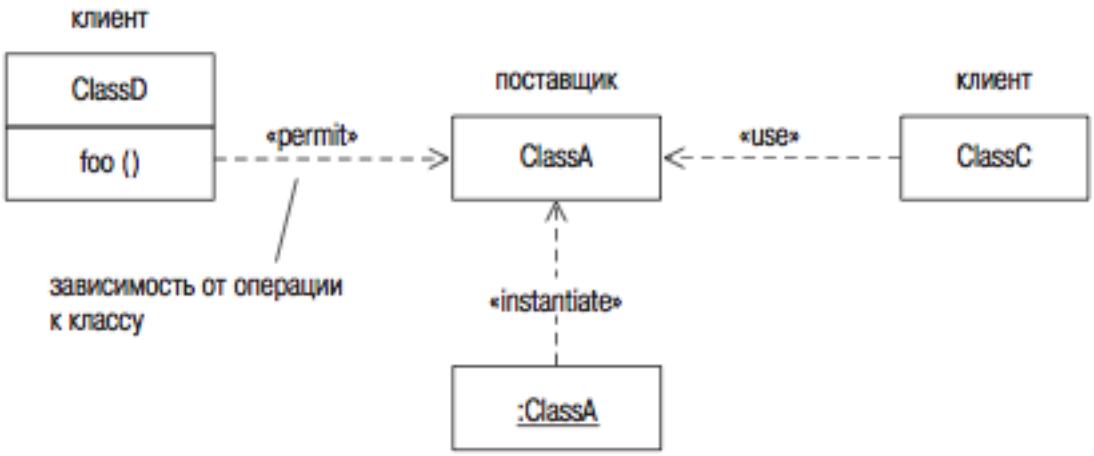


Рис. 5.6. Типы зависимостей

5.1.6. Обобщение

Обобщение – это отношение между более общей сущностью и более специальной сущностью.

Два элемента подчиняются принципу замещаемости: более специальный элемент может использоваться *везде,* где предполагается использование более общего элемента, без нарушения системы. Очевидно, что обобщение – намного более прочный тип отношений, чем ассоциация. В самом деле, обобщение подразумевает самый высокий уровень зависимости (и, следовательно, связанности) между двумя элементами.

5.1.7.Наследование классов

В иерархии обобщения кроется наследование между классами, посредством которого подклассы наследуют все возможности своих надклассов. Чтобы быть более специальными, подклассы наследуют:

- атрибуты;
- операции;
- отношения;
- ограничения.

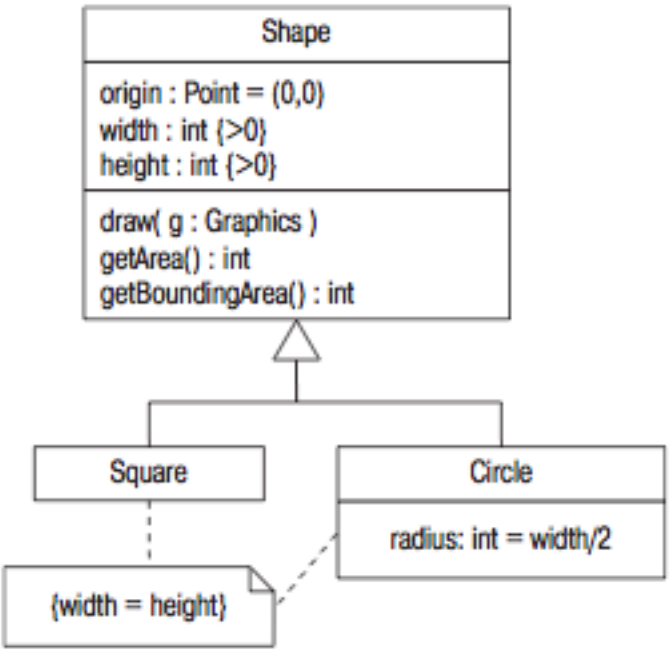


Рис. 5.7. Наследование характеристик суперкласса

5.1.8.Интерфейсы

Интерфейс определяет именованный набор открытых свойств.

Главная идея, лежащая в основе интерфейсов, – разделение *описания* функциональности (интерфейс) от ее *реализации* классификатором, таким как класс или подсистема. Создать экземпляр интерфейса невозможно. Он просто объявляет контракт, который может быть реализован классификаторами. Все, что реализует интерфейс, принимает и соглашается следовать определяемому интерфейсом контракту.

В интерфейсах также должны присутствовать описания семантики их возможностей (обычно в виде текста или псевдокода) как руководства для тех, кто будет их реализовывать.

Важно помнить, что интерфейс определяет только *описание* своих возможностей и что он *никогда* не заключает в себе какой-либо конкретной реализации.

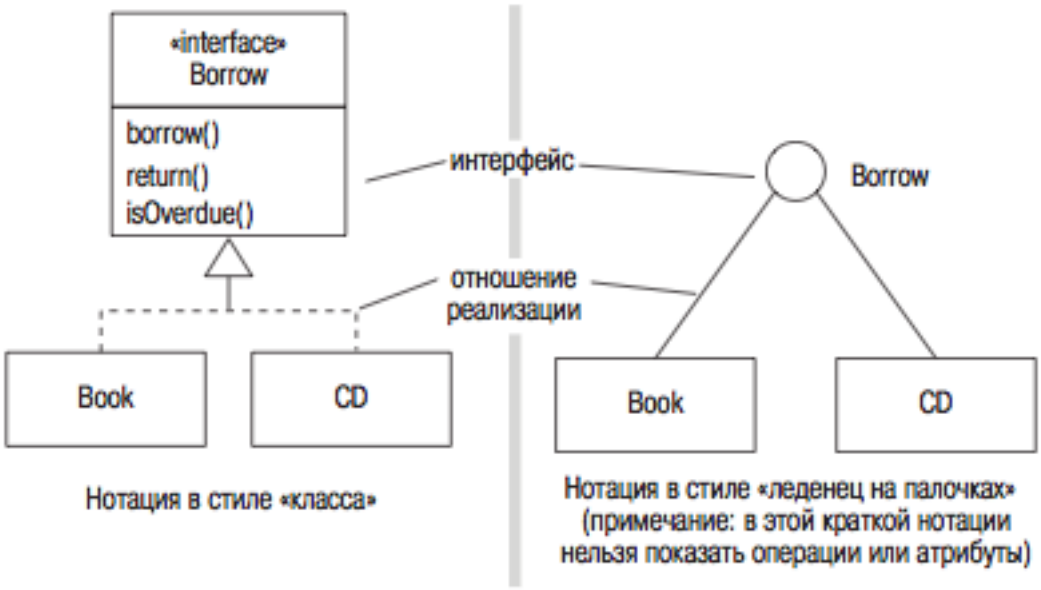


Рис. 5.8. Два варианта синтаксиса для предоставляемых интерфейсов

5.1.9.Порты

Порт группирует семантически связанный набор предоставляемых и требуемых интерфейсов. Он указывает на конкретную точку взаимодействия классификатора и его окружения.

Порты являются очень удобным способом структурирования предоставляемых и требуемых интерфейсов классификатора. Их также можно использовать для упрощения диаграммы. Например, на рис. 5,9 показан класс Viewer (средство просмотра), соединяющийся с портом presentation класса Book. Чтобы обеспечить возможность соединения портов, их предоставляемый и требуемый интерфейсы должны

совпадать. Использование портов, очевидно, обеспечивает намного более краткое представление, чем отображение всех предоставляемых и требуемых интерфейсов, но может и усложнять чтение диаграмм.

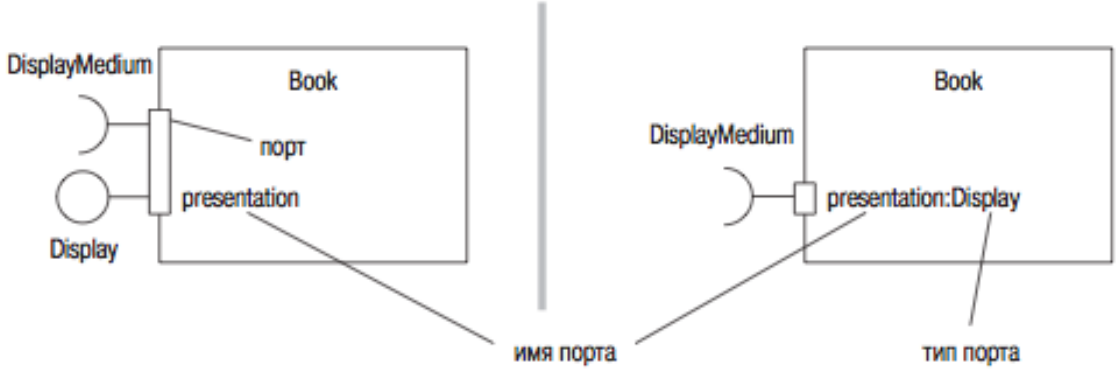


Рис. 5.9. Два варианта нотации порта

5.2.Задание на лабораторную работу.

- Изучить теоретическую часть лабораторной работы.
- Для произвольной программной системы сформировать статическую структуру классов, используя различные средства для представления взаимодействия классов.
- Для разработанной структуры классов сформировать структуру базы данных в нотации диаграммы классов.
- Для разработанной структуры классов сформировать пользовательский интерфейс в нотации диаграммы классов.
- Оформить отчёт по лабораторной работе

5.3.Требования к содержанию отчёта

Отчёт по лабораторной работе должен содержать:

- Название работы
- Цель работы
- Краткое описание проектируемой системы
- Структура программного модуля в виде диаграмма классов
- Структура базы данных в нотации диаграммы классов
- Структура пользовательского интерфейса в нотации диаграммы классов
- Выводы по работе

6.ЛАБОРАТОРНАЯ РАБОТА № 4. МОДЕЛИРОВАНИЕ ДИНАМИЧЕСКОЙ СТРУКТУРЫ ПРОГРАММНОЙ СИСТЕМЫ (ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТЕЙ И КОММУНИКАЦИОННАЯ ДИАГРАММА).

Цель работы: Целью работы является построение динамической модели поведения программной системы посредством построения диаграмм взаимодействия (диаграмма последовательностей и коммуникационная диаграмма).

6.1.Диаграммы взаимодействий

Диаграммы взаимодействий UML2 могут использоваться для моделирования любого типа взаимодействия между экземплярами классификаторов. В частности, в реализации прецедентов диаграммы взаимодействий используются для моделирования взаимодействий между объектами, реализующими прецедент или его часть. Существует четыре разных типа диаграмм взаимодействий, каждый из которых делает акцент на различных аспектах взаимодействия.

Диаграммы последовательностей (sequence diagrams) акцентируют внимание на временной упорядоченности сообщений. Обычно пользователи лучше понимают диаграммы последовательностей, чем коммуникационные диаграммы, поскольку они намного легче читаются. Как правило, коммуникационные диаграммы очень быстро загромождаются.

Коммуникационные диаграммы (communication diagrams) выделяют структурные отношения между объектами и очень полезны при анализе, особенно для создания эскиза совместной работы объектов. В UML2 эти диаграммы предлагают только лишь подмножество функциональности диаграмм последовательностей.

Диаграммы обзора взаимодействий (interaction overview diagrams) показывают, как сложное взаимодействие реализуется рядом простых взаимодействий. Это особый случай диаграммы деятельности, в которой узлы ссылаются на другие взаимодействия. Они полезны для моделирования потока управления системы.

Временные диаграммы (timing diagrams) обращают внимание на фактическое время взаимодействия. Их основное назначение – по□ мочь оценить временные затраты.

6.2.Диаграммы последовательностей

Диаграммы последовательностей представляют взаимодействия между линиями жизни как упорядоченную последовательность событий. Это самая богатая и гибкая форма диаграммы взаимодействий.

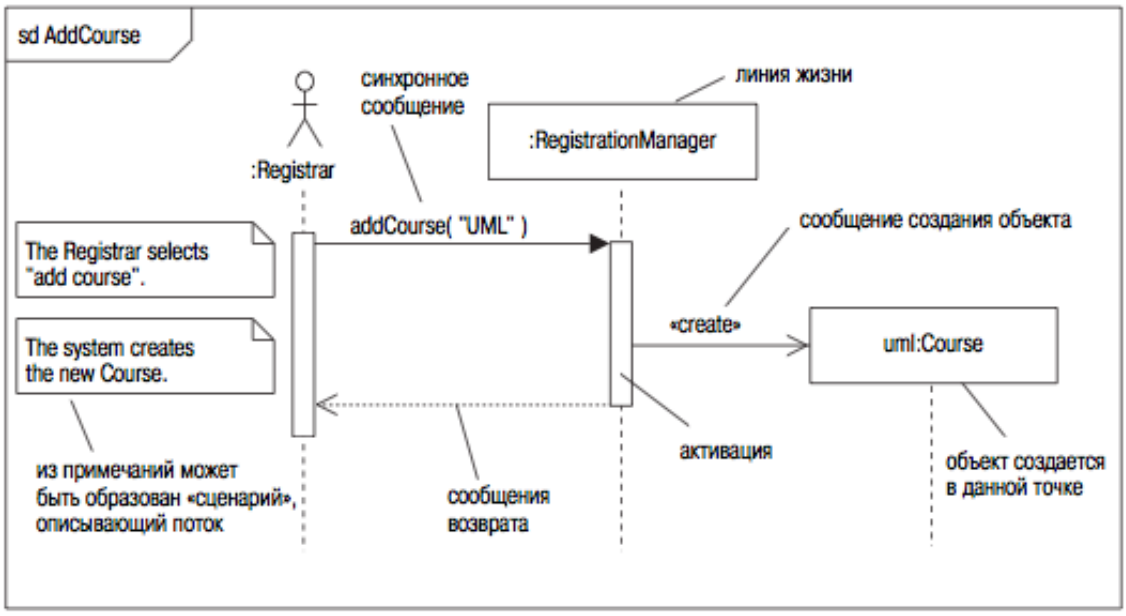


Рис. 6.1. Диаграмма последовательностей прецедента AddCourse

На рис. 6.1. показана диаграмма последовательностей, реализующая поведение, описанное прецедентом AddCourse. Согласно спецификации UML2, имена диаграмм взаимодействий могут начинаться с приставки sd, для обозначения того, что данная диаграмма является диаграммой взаимодействий. Довольно странно: sd используется для *всех* типов диаграмм взаимодействий, а не только для диаграмм последовательностей (sd – sequence diagram)!

Время на диаграммах последовательностей развивается сверху вниз, линии жизни выполняются слева направо. Линии жизни размещаются горизонтально, чтобы минимизировать число пересекающихся линий на диаграмме. Их местоположение относительно вертикальной оси отражает момент их создания. Пунктирная линия, находящаяся внизу, показывает существование линии жизни в течение времени.

Вытянутые прямоугольники, расположенные на пунктирной линии под линией жизни, показывают, когда на данной линии жизни

находится фокус управления. Эти прямоугольники называются *активацями,* или *фокусом управления.*

6.3.Коммуникационные диаграммы

Коммуникационные диаграммы акцентируют внимание на структурных аспектах взаимодействия, на том, как соединяются линии жизни. В UML2 их семантика довольно слаба по сравнению с диаграммами последовательностей.

Коммуникационные диаграммы акцентируют внимание на структурных аспектах взаимодействия.

Синтаксис коммуникационных диаграмм подобен синтаксису диаграмм последовательностей, за исключением того, что здесь у линий жизни нет «хвостов». Вместо этого они соединены связями, образующими коммуникационные каналы для передачи сообщений. Порядок сообщений определяется путем иерархической нумерации.

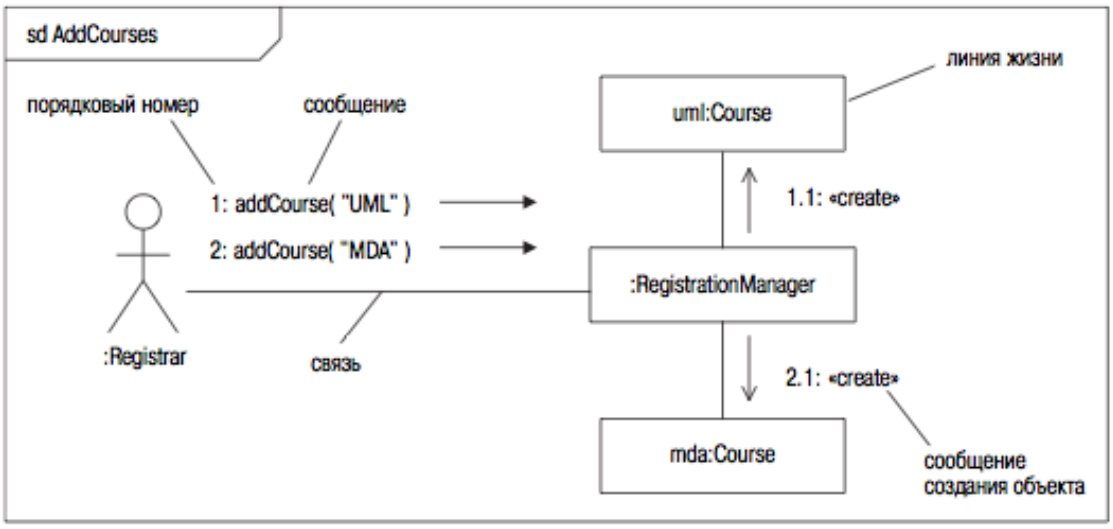


Рис. 6.2. Коммуникационная диаграмма прецедента AddCourses

Главное необходимо понимать, что в результате отправления сообщения вызывается некоторая операция экземпляра и что сложная нумерация сообщений указывает на вложенность вызовов операций (т. е. вложенный фокус управления).

6.4.Задание на лабораторную работу.

- Изучить теоретическую часть лабораторной работы.
- Для произвольной программной системы разработать временную диаграмму поведения системы

- Для произвольной программной системы разработать диаграму развертывания системы
- Оформить отчет по лабораторной работе

7.ЛАБОРАТОРНАЯ РАБОТА № 5. МОДЕЛИРОВАНИЕ ДИНАМИЧЕСКОЙ СТРУКТУРЫ ПРОГРАММНОЙ СИСТЕМЫ (ДИАГРАММА ДЕЯТЕЛЬНОСТИ И ДИАГРАММА ОБЗОРА ВЗАИМОДЕЙСТВИЯ)

Цель работы: Целью работы является построение динамической модели поведения программной системы посредством построения диаграмм взаимодействия (диаграмма деятельности и диаграмма обзора взаимодействия).

7.1.Диаграммы деятельности

В UML2 диаграммы деятельности фактически были лишь особым случаем диаграмм состояний, где у каждого состояния было входное действие, которое определяло некоторый процесс или функцию, имеющие место при входе в состояние. В UML2 диаграммы деятельности имеют совершенно новую семантику, базирующуюся на технологии сетей Петри (Petri Nets). В использовании этой технологии есть два преимущества:

1. Формализм сети Петри обеспечивает большую гибкость при моделировании различных типов потока.

2. В UML2 теперь есть четкое разделение между диаграммами деятельности и диаграммами состояний.

Диаграммы деятельности – это объектно ориентированные блок схемы.

Деятельность может быть добавлена к *любому* элементу модели с целью моделирования его поведения. Элемент обеспечивает контекст для деятельности, и деятельность может использовать возможности своего контекста.

Диаграммы деятельности также могут использоваться для моделирования бизнес процессов и рабочих потоков.

Хорошая диаграмма деятельности сосредоточена на отражении лишь одного определенного аспекта динамического поведения системы. Таким образом, она должна находиться на соответствующем уровне абстракции, чтобы донести эту идею до целевой аудитории, и содержать минимум необходимой информации.

Деятельности – это системы *узлов* (nodes), соединенных *ребрами* (edges). Существует три категории узлов:

- **Узлы действия (action nodes)** – представляют отдельные единицы работы, элементарные *в рамках деятельности;*

- **Узлы управления (controlnodes)** –управляют потоком деятельности;
- **Объектные узлы (object nodes)** – представляют объекты, используемые в деятельности.

Ребра представляют потоки деятельности. Существует два типа ребер:

1. **Ребра потоков управления (control flows)** – представляют поток управления деятельности;

2. **Ребра потоков объектов (object flows)** – представляют поток объектов деятельности.

Деятельности обычно начинаются с одного узла управления, начального. Он обозначает начало исполнения при вызове деятельности. Один или более конечных узлов показывают места завершения деятельности.

Обычно диаграммы деятельности используются для моделирования прецедента в виде последовательностей действий.

7.1.1.Узлы действия

Существует 4 типа узлов действия, перечисленных в табл. 7.1.

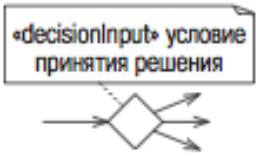
Таблица 7.1.

Синтаксис	Имя	Семантика
	Узел вызова действия	Инициирует деятельность, поведение или операцию.
	Посылка сигнала	Действие посылки сигнала – посылает сигнал асинхронно (отправитель *не* ожидает подтверждения получения сигнала). Для создания сигнала может принимать входные параметры.
	Узел действия, принимающий событие	Принимает событие – ожидает события, установленного объектом-владельцем, и выдает событие на выходе. Активируется при получении маркера по входящему ребру. Если *нет* входящего ребра, запускается при запуске включающей его деятельности и всегда является активированным.
	Узел действия, принимающий событие времени	Принимает событие времени – отвечает на определенное значение времени. Генерирует события времени соответственно своему временному выражению.

7.1.2.Узлы управления

Узлы управления контролируют поток управления деятельности.

Таблица 7.2

Синтаксис	Имя	Семантика	
	Начальный узел	Указывает, где начинается поток при вызове деятельности.	
	Конечный узел деятельности	Завершает деятельность.	Конечные узлы
	Конечный узел потока	Завершает определенный поток деятельности – другие потоки не затрагиваются.	
	Узел решения	Поток проходит по исходящему ребру, сторожевое условие которого истинно. Может иметь входные данные (необязательно).	
	Узел слияния	Копирует входные маркеры в единственное выходное ребро.	
	Узел ветвления	Разделяет поток на несколько параллельных потоков.	
	Узел объединения	Синхронизирует несколько параллельных потоков. Может иметь описание объединения (не обязательно) для изменения его семантики.	

7.1.3.Диаграмма деятельности

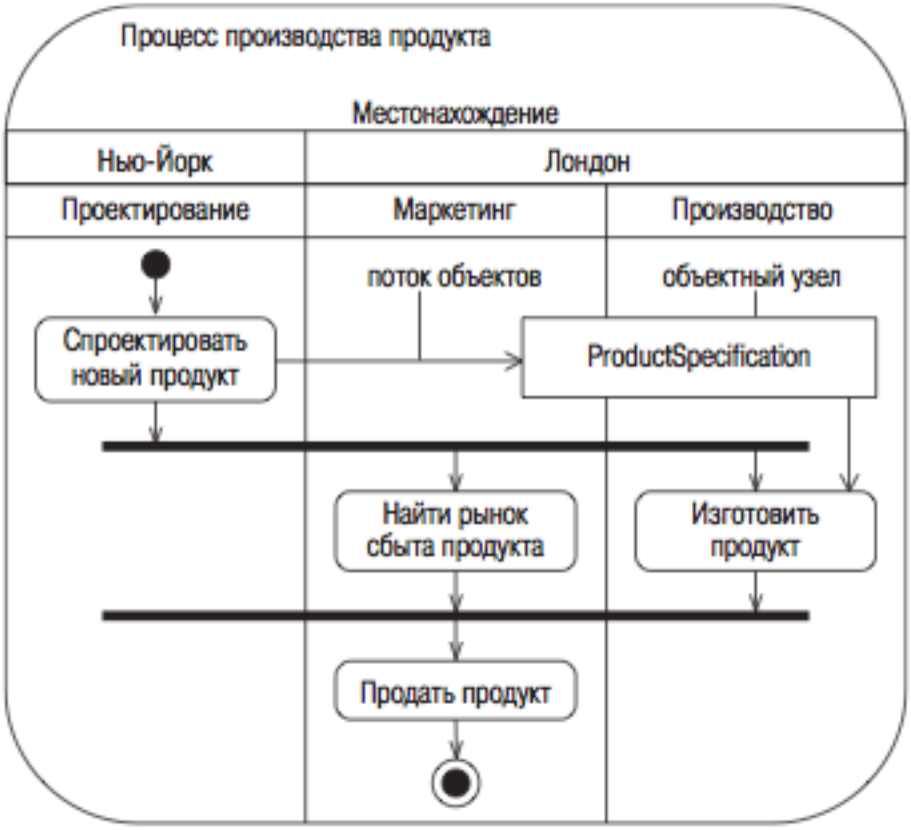


Рис. 7.1. Деятельность

7.1.4.Разъёмы

Основное правило: применения разъёмов необходимо избегать. Однако если диаграмма очень сложна и не поддаётся упрощению, разъёмы можно использовать для разрыва длинных рёбер, которые трудно проследить, и для «распутывания» пересекающихся рёбер. Это может упростить диаграммы деятельности и повысить их удобочитаемость.

Синтаксис разъёма представлен на рис. 7.2. В заданной деятельности каждому исходящему разъёму *должен* соответствовать единственный входящий разъем с такой же меткой. Метки – это идентификаторы разъёма, никакой другой семантики у них нет. Обычно в их качестве выступают буквы алфавита.

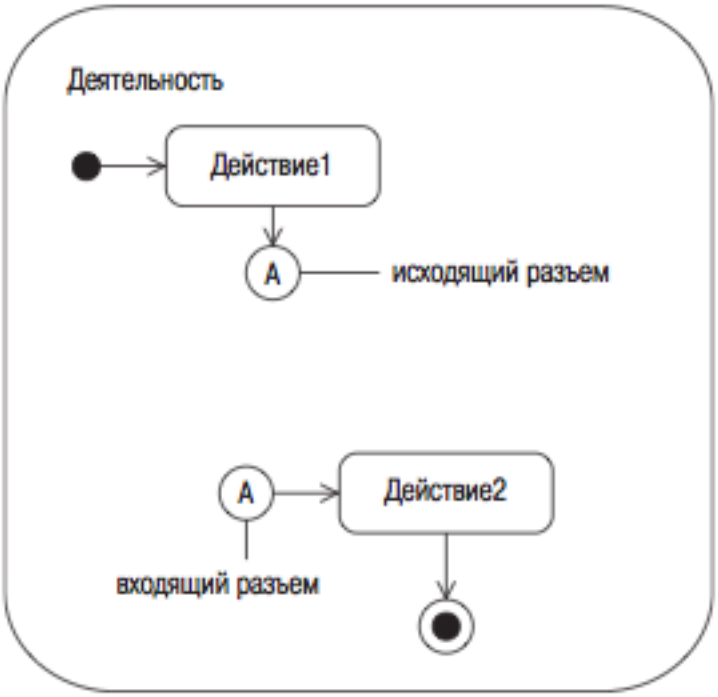


Рис. 7.2. Синтаксис разъёма

7.1.5.Области с прерываемым выполнением действий

Области с прерываемым выполнением действий – это области деятельности, прерываемые при прохождении маркера по прерывающему ребру. Когда область прерывается, все ее потоки немедленно прекращаются. Области с прерываемым выполнением действий полезны для моделирования прерываний и асинхронных событий. Чаще всего они используются при проектировании, но также могут быть полезны при анализе для представления процесса обработки асинхронных бизнес событий.

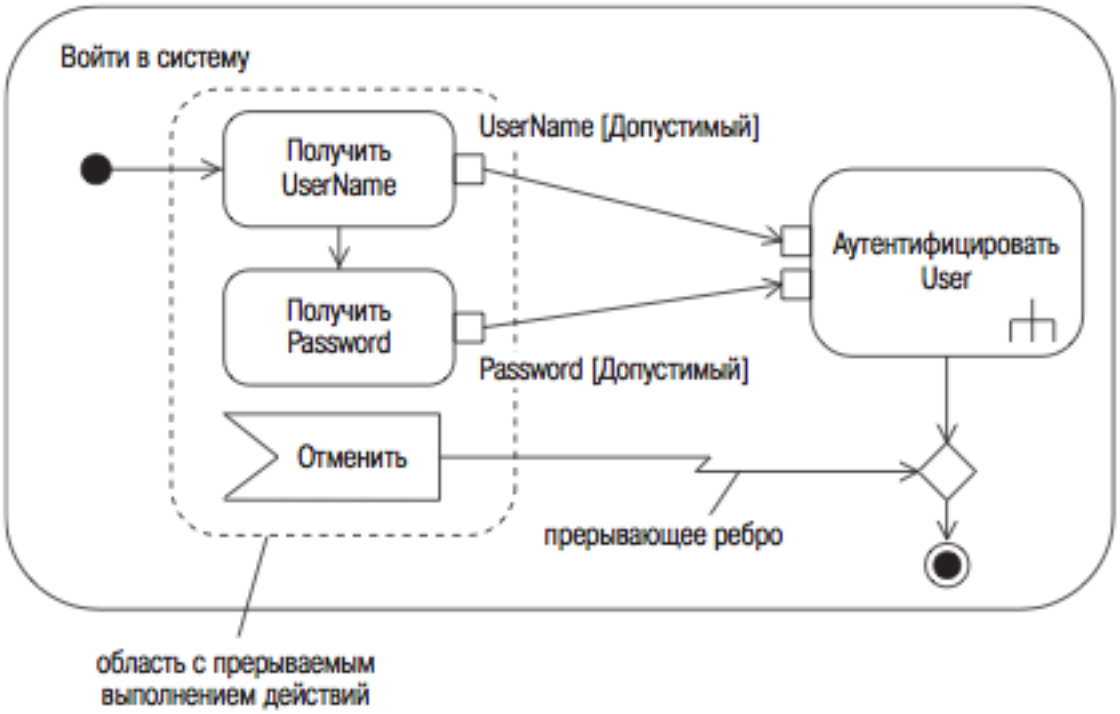


Рис. 6

7.3. Деятельность. Вход в систему имеет область с прерываемым выполнением действий

7.1.6.Обработка исключений

Современные языки программирования часто обрабатывают ошибки посредством механизма, называемого *обработкой исключений* (excep*tion handling)*. В случае выявления ошибки в защищенной части кода создается объект исключения. Поток управления переходит в обработчик исключения, который некоторым образом обрабатывает объект исключения. В этом объекте исключения содержится информация об ошибке, которая может использоваться обработчиком исключения. Обработчик исключения может прервать приложение или попытаться восстановить нормальное состояние. Часто информация объекта исключения сохраняется в журнале регистрации ошибок.

Обработку исключений на диаграммах деятельности можно моделировать с помощью контактов исключений, защищенных узлов и обработчиков исключений.

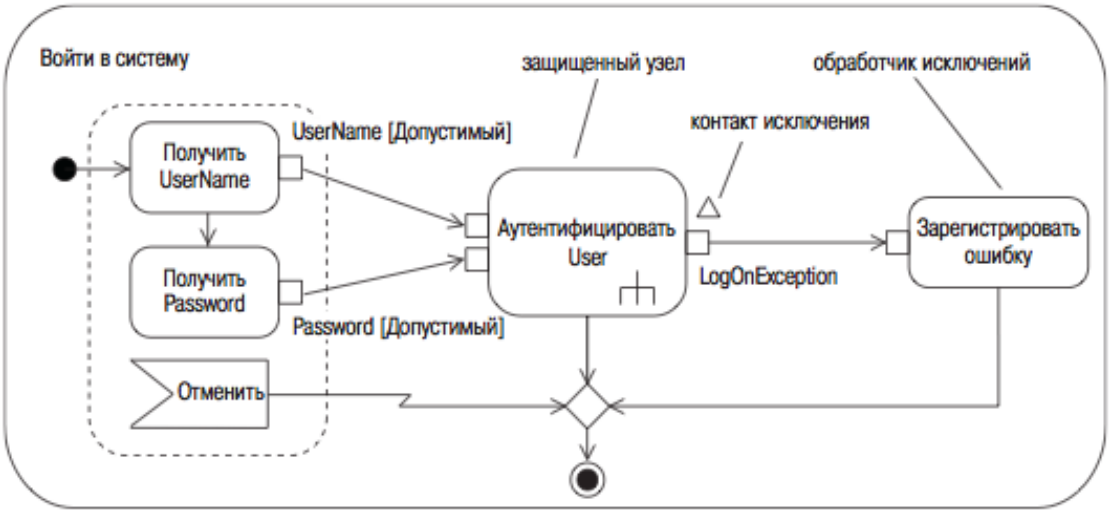


Рис. 7.4. Деятельность. Вход в систему с обработчиком исключений

Для того чтобы показать, что выходной контакт представляет собой объект исключения, его помечают небольшим равносторонним треугольником (рис. 7.4). Узел Зарегистрировать ошибку выступает в роли обработчика исключений, генерируемых действием Аутентифицировать User. Если с узлом ассоциирован обработчик исключений, узел называют защищенным.

7.1.7.Узлы расширения

Узел расширения – это объектный узел, который представляет коллекцию объектов, входящую в или выходящую из области расширения. Область расширения исполняется по одному разу для каждого входного элемента. На рис. 7.5 показан пример области расширения. Она представлена в виде отрисованного пунктиром прямоугольника со скругленными углами и входящим и исходящим узлами расширения. Узел расширения выглядит как контакт, но с тремя квадратными ячейками. Это указывает на поступление коллекции, а не одного объекта.

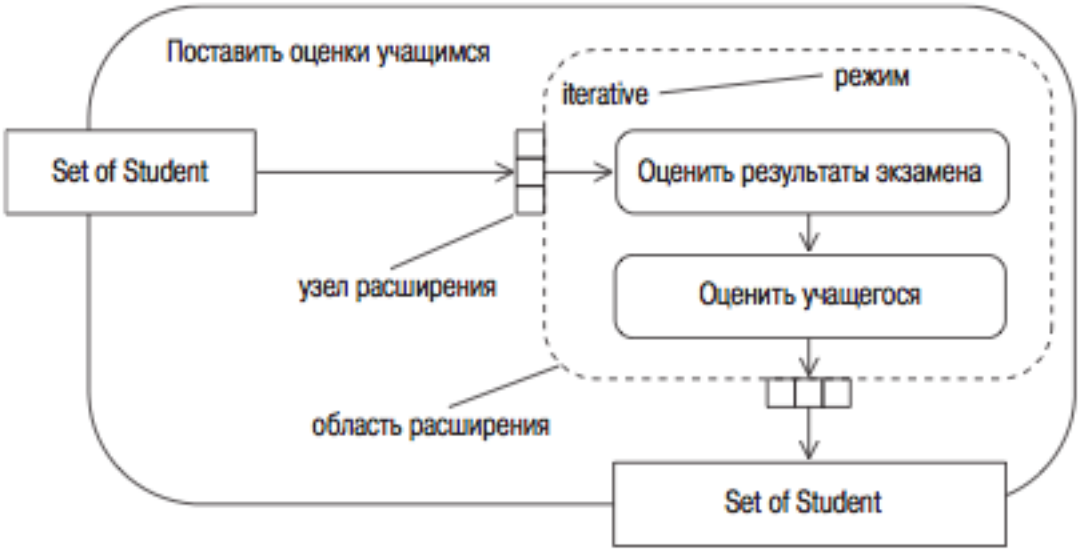


Рис. 7.5. Пример области расширения

На узлы расширения накладываются два ограничения.

- Тип выходной коллекции *должен* соответствовать типу входной коллекции.
- Типы объектов входной и выходной коллекций *должны* быть одинаковыми. Эти ограничения означают, что области расширения не могут использоваться для преобразования входных объектов одного типа в выходные объекты другого типа. Количество выходных коллекций может не совпадать с количеством входных, поэтому области расширения могут использоваться для объединения или разделения коллекций.

7.1.8.Отправка сигналов и приём событий

Сигнал представляет асинхронно передаваемую между объектами информацию. Сигнал моделируется как класс, отмеченный стереотипом «signal» (сигнал). Передаваемая информация хранится в атрибутах сигнала. При анализе сигналы могут использоваться для отображения отправки и получения асинхронных бизнес событий (таких как OrderReceived (заказ получен)), а при проектировании они могут иллюстрировать асинхронный обмен информацией между разными системами, подсистемами или частями оборудования.

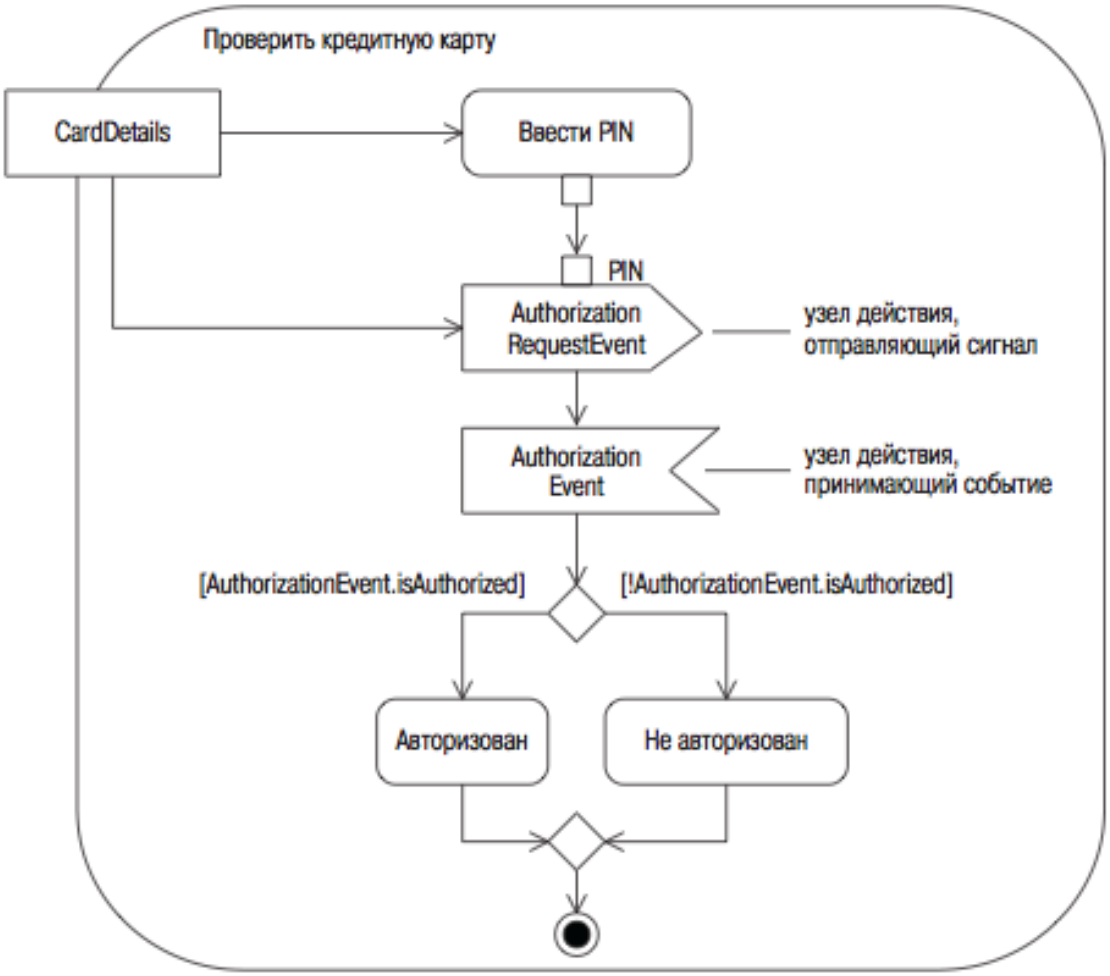


Рис. 7.6. Деятельность Проверить кредитную карту

Сигнал можно послать с помощью узла действия, отвечающего за отправку сигнала. Он посылает сигнал асинхронно – деятельность, отправляющая сигнал, *не* ожидает подтверждения получения сигнала.

7.1.9.Потоковая передача

Обычно действия принимают маркеры со своих входных ребер только тогда, когда начинают выполнение, и предлагают их на своих выходных ребрах по завершении выполнения. Однако иногда требуется, чтобы действие выполнялось *непрерывно,* периодически принимая и предлагая маркеры. Такое поведение называют потоковой передачей. В UML2 оно может быть проиллюстрировано любым из четырех способов, как показано на рис. 7.7.

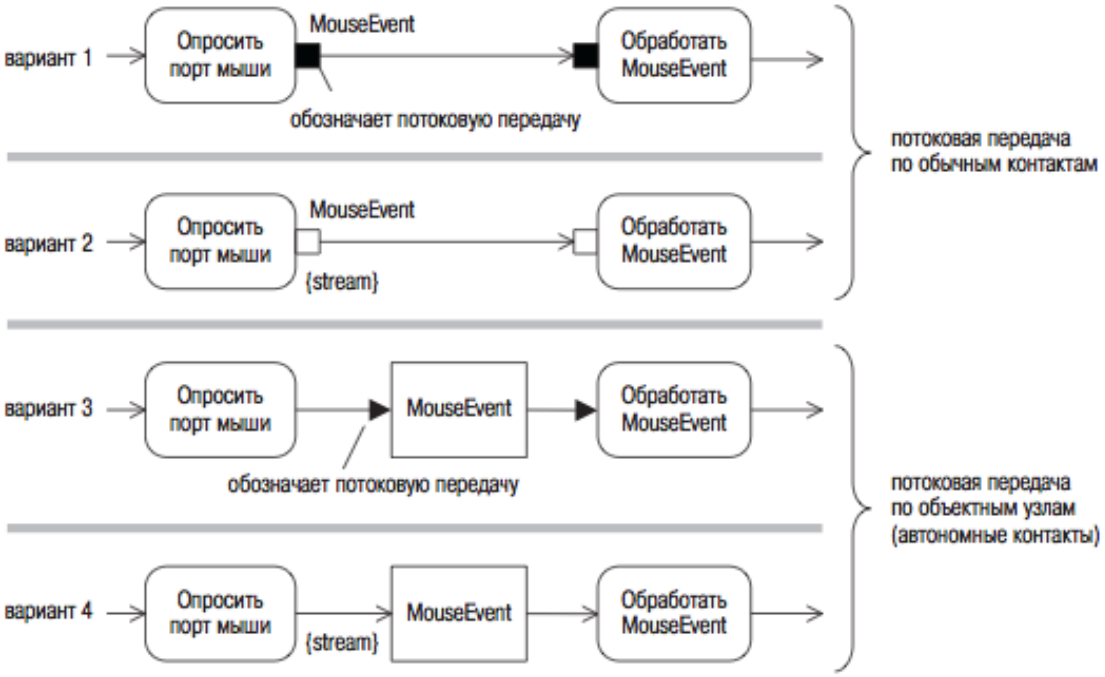


Рис. 7.7. Четыре способа отображения потоковой передачи

Любая ситуация, в которой необходимо *непрерывно* получать и обрабатывать информацию, является кандидатом на использование потоковой передачи.

7.2.Диаграммы обзора взаимодействий

Диаграммы обзора взаимодействий – особая форма диаграммы деятельности. Они показывают взаимодействия и включения взаимодействий (interaction occurrences) и используются для моделирования высокоуровневого потока управления между взаимодействиями.

Одно особенно мощное применение диаграмм обзора взаимодействий – иллюстрация потока управления между прецедентами. Если каждый прецедент представить как взаимодействие, синтаксис диаграммы деятельности можно использовать для отображения движения потока управления между ними.

Обратите внимание, что линии жизни, принимающие участие во взаимодействии, могут быть перечислены после ключевого слова lifelines (линии жизни) в заголовке диаграммы. Такое документирование может быть полезным, поскольку часто линии жизни скрыты внутри включений взаимодействий.

Синтаксис диаграмм обзора взаимодействий аналогичен синтаксису диаграмм деятельности, за исключением того, что здесь отображаются встроенные взаимодействия и включения взаимодействий, а не деятельности и объектные узлы.

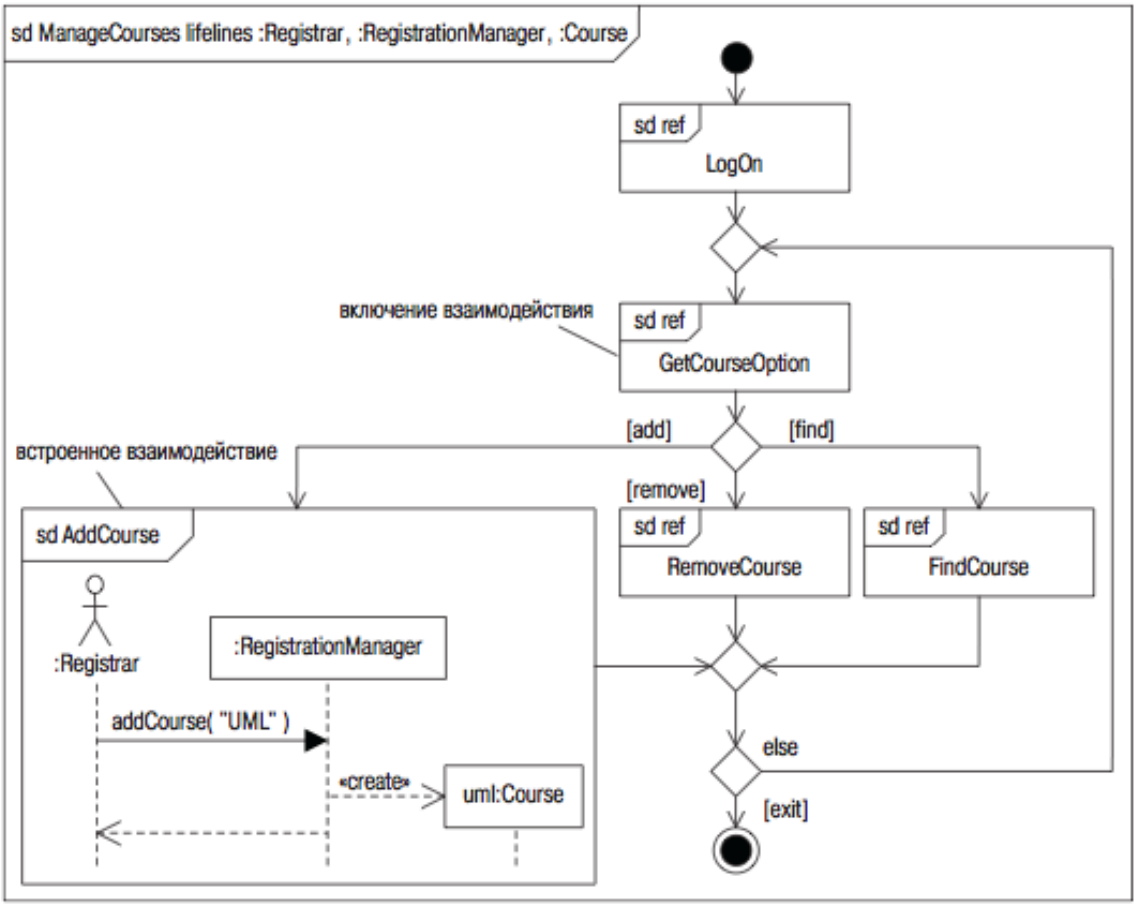


Рис. 7.8. Диаграмма обзора взаимодействий

Диаграммы обзора взаимодействий дают визуальное представление ветвления, параллелизма и итерации, что намного понятней и проще для восприятия. В то же время они переносят ваше внимание с элементов, например линий жизни, непосредственно на поток управления. Поэтому эти диаграммы должны использоваться, когда необходимо акцентировать внимание на движении потока управления через множество взаимодействий.

В реализации прецедента взаимодействия описывают поведение, определенное в прецеденте. Таким образом, диаграммы обзора взаимодействий могут использоваться для иллюстрации бизнес процессов, охватывающих прецеденты.

7.3.Задание на лабораторную работу.

- Изучить теоретическую часть лабораторной работы.
- Для произвольной программной системы описать динамику поведения программной системы.
- Обормить структуру программ в виде динамических диаграмм.
- Оформить отчет по лабораторной работе

8.ЛАБОРАТОРНАЯ РАБОТА № 6. МОДЕЛИРОВАНИЕ ДИНАМИЧЕСКОГО ПОВЕДЕНИЯ СИСТЕМЫ (ДИАГРАММА СОСТОЯНИЙ)

Цель работы: Целью работы является построение модели динамического поведения системы посредством построения диаграммы состояний (конечных автоматов).

8.1.Конечные автоматы

Конечный автомат моделирует динамическое поведение реактивного объекта.

- Спецификация UML2 определяет два типа конечных автоматов, имеющих общий синтаксис:
  - поведенческие автоматы;
  - протокольные автоматы.

Поведенческие автоматы с помощью состояний, переходов и событий определяют *поведение* контекстного классификатора. Они могут использоваться, только если у контекстного классификатора есть некоторое поведение, которое можно смоделировать. У некоторых классификаторов, например интерфейсов и портов, такого поведения нет; они просто описывают протокол использования. Состояния поведенческих автоматов могут определять одно или более действий, выполняемых при входе в состояние, нахождении в нем или выходе из него.

Протокольные автоматы используют состояния, переходы и события для определения *протокола* контекстного классификатора. Протокол включает:

- условия, при которых могут вызываться операции классификатора и его экземпляров;
- результаты вызовов операций;
- порядок вызовов операций.

Протокольные автоматы ничего не сообщают о реализации этого поведения. Они только определяют, как оно представляется внешней сущности. Протокольные автоматы могут использоваться при описании протокола для *всех* классификаторов, включая те, которые не имеют реализации. Состояния протокольных автоматов *не могут* определять действия; это дело поведенческих автоматов.

8.2.Диаграммы состояний

Для иллюстрации диаграмм состояний (state machine diagrams) давайте рассмотрим простой пример. Один из самых наглядных

объектов реального мира, который постоянно переходит из состояния в состояние, – электрическая лампочка. На рис. 8.1 показана передача событий от переключателя к лампочке. Могут быть посланы два события: turnOn (включить) (это событие моделирует подключение лампы в электрическую сеть) и turnOff (выключить) (выключает ток).

Диаграмма состояний содержит только один конечный автомат для единственного реактивного объекта. В данном случае реактивный объект – это система, состоящая из лампочки, переключателя и электропитания. Диаграмма состояний может изображаться в явно обозначенной рамке, как показано на рис. 8.1, или существовать в неявных рамках, предоставляемых средством моделирования.

У каждого конечного автомата должно быть начальное состояние (закрашенный кружок), обозначающее первое состояние последовательности. Если смена состояний не бесконечна, должно присутствовать и конечное состояние (бычий глаз), которое завершает последовательность переходов. Обычно переход от начального псевдосостояния к первому «настоящему» состоянию происходит автоматически. Начальное псевдосостояние используется просто как удобный маркер для обозначения начала ряда переходов состояний.

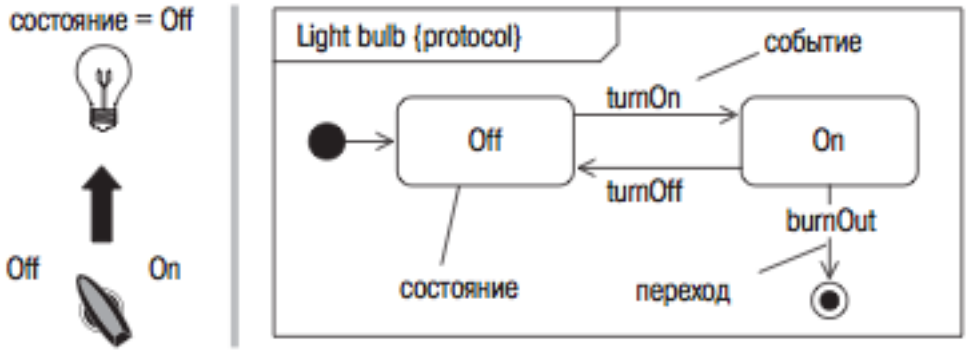


Рис. 8.1. Диаграмма состояний электрической лампочки

8.2.1.Синтаксис состояния

UML2 синтаксис состояния представлен на рис. 8.2.

Каждое состояние поведенческого автомата может содержать нуль или более действий и деятельностей. У состояний протокольных автоматов *нет* действий или деятельностей.

Действия считаются мгновенными и непрерываемыми, тогда как деятельности занимают конечное время и могут быть прерваны. Каждое действие в состоянии ассоциируется с внутренним переходом,

инициируемым событием. В состоянии может быть любое число действий и внутренних переходов.

Внутренний переход позволяет зафиксировать тот факт, что произошло что-то, заслуживающее отражения в модели, но не обуславливающее (или не настолько важное, чтобы моделировать это как) переход в новое состояние. Например, на рис. 8.2 нажатие одной из клавиш клавиатуры, конечно, является заслуживающим внимания событием, но оно не приводит к переходу из состояния EnteringPassword (ввод пароля). Мы моделируем это как внутреннее событие keypress (нажатие клавиши), которое обусловливает внутренний переход, инициирующий действие отобразить “*“.

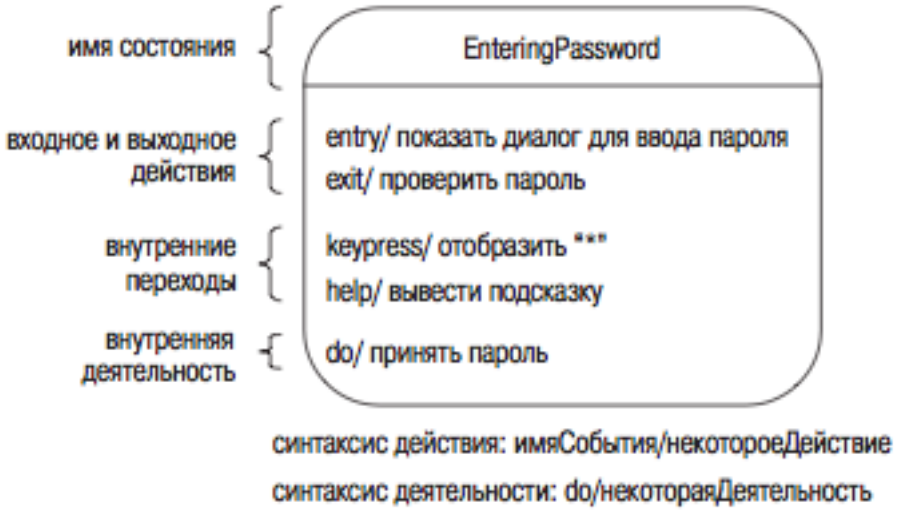


Рис. 8.2. Синтаксис состояния

Два специальных действия – вход и выход – ассоциированы со специальными событиями entry и exit. У этих двух событий особая семантика. Событие entry происходит мгновенно и автоматически при входе в состояние. Это первое, что происходит, когда осуществляется вход в состояние. Это событие обусловливает выполнение ассоциированного с ним действия на входе. Событие exit – самое последнее, что происходит мгновенно и автоматически при выходе из состояния. Обусловливает выполнение ассоциированного действия на выходе.

8.2.2. Переходы

UML2 синтаксис переходов для поведенческих автоматов представлен на рис. 8.3.

Переходы показывают движение между состояниями.

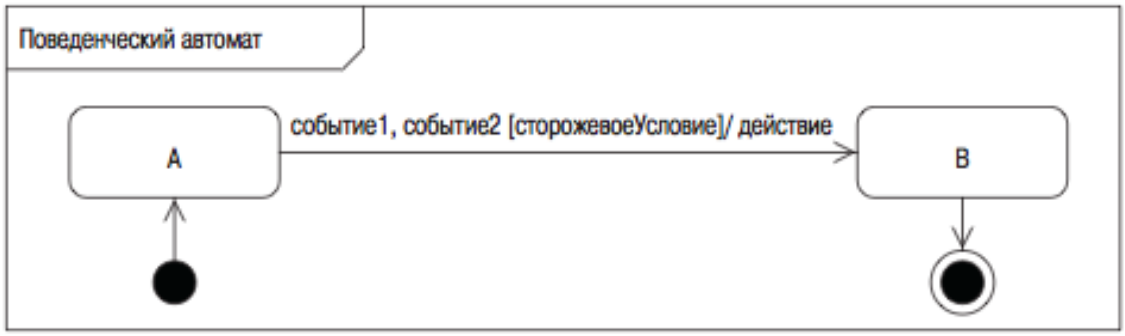


Рис. 8.3. Синтаксис переходов для поведенческих автоматов

Синтаксис переходов в поведенческом автомате прост и может использоваться для внешних переходов (изображаются стрелками) или внутренних переходов (изображаются вложенными в состояние). Каждый переход имеет три необязательных элемента.

1. Нуль или более событий – определяют внешнее или внутреннее происшествие, которое может инициировать переход.
2. Нуль или одно сторожевое условие – логическое выражение, которое должно быть выполнено (иметь значение true), прежде чем может произойти переход. Условие указывают после событий.
3. Нуль или более действий – часть работы, ассоциированная с переходом и выполняемая при срабатывании перехода.

8.2.3.События

События инициируют переходы. UML2 определяет событие как «описание заслуживающего внимания происшествия, занимающего определенное положение во времени и пространстве». События инициируют переходы в автоматах. События могут быть указаны вне состояний на переходах, или внутри состояний.

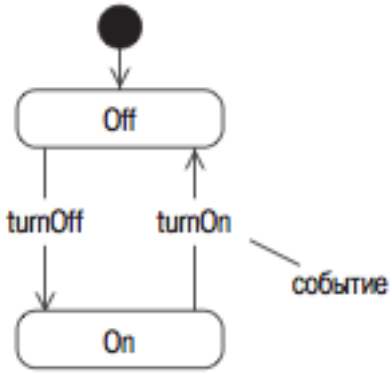


Рис. 8.4. События указаны вне состояний на переходах

Существует четыре семантически различных типа событий:

- событие вызова;
- сигнал;
- событие изменения;
- событие времени.

8.2.4.Составные состояния

Составное состояние – это состояние, содержащее вложенные состояния.

Эти вложенные состояния объединяются в один или более конечных автоматов, которые называют *подавтоматами* (submachines). Для каждого подавтомата в пиктограмме композиции отведена собственная *область*. Области – это просто участки пиктограммы состояния, разделенные пунктирными линиями.

Рис. 8.5. Каждому подавтомату отведена своя область

Вложенные состояния также могут быть составными состояниями. Однако, как правило, необходимо по возможности стремиться к тому, чтобы вложенность составных состояний не превышала двух или трех уровней. При большей глубине вложенности автомат сложно воспринимать на диаграмме и понимать.

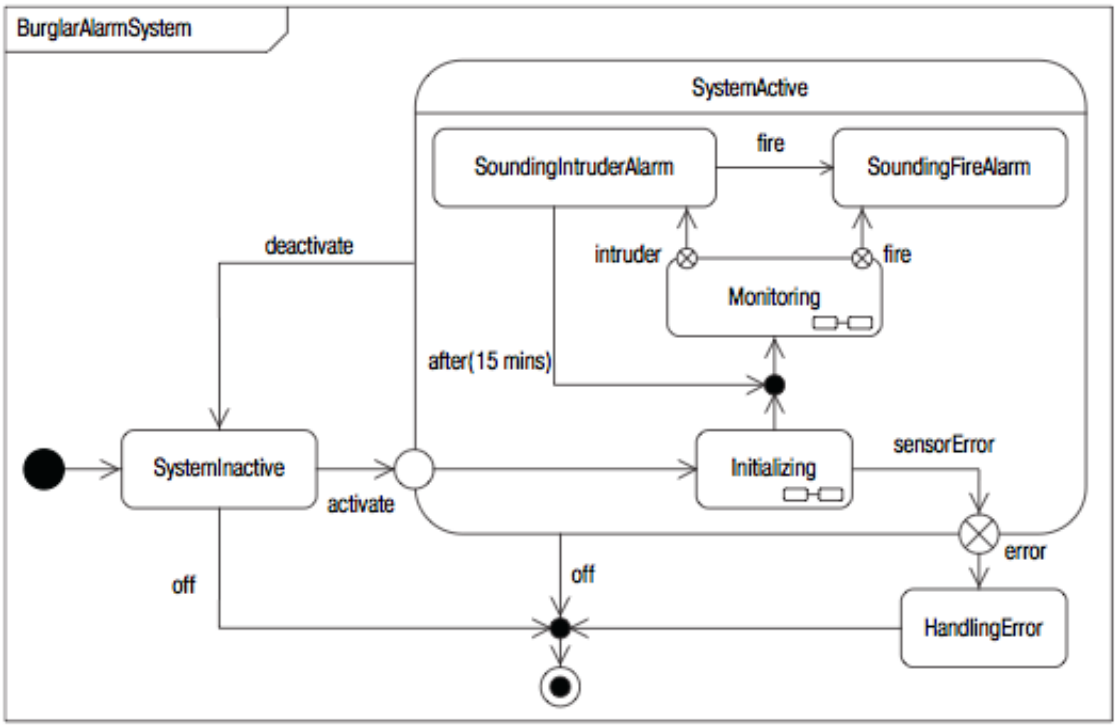


Рис. 8.6. Автомат системы сигнализации

При входе в это состояние происходит ветвление и начинается параллельное выполнение двух подавтоматов. В верхнем подавтомате состояние InitializingFireSensors выполняет процесс инициализации пожарных датчиков. В нижнем подавтомате состояние InitializingSecuritySensors делает то же самое для датчиков безопасности.

В нормальных условиях по завершении *обоих* подавтоматов происходит автоматический выход из суперсостояния Initializing. Это объединение: подавтоматы синхронизируются таким образом, что дальнейшая работа невозможна, пока не будут инициализированы *и* пожарные датчики, *и* датчики безопасности.

8.3.Временные диаграммы

Одним из слабых мест UML2 было моделирование систем реального времени. Это такие системы, в которых временные соотношения критически важны и события должны следовать друг за другом в рамках определенного временного окна. Мы говорим «временное окно», а не «время», потому что абсолютное время для нас как разработчиков *неприемлемо*. Когда в модели задается время, на

самом деле задается время плюс-минус некоторая погрешность, определяемая внешними факторами, такими как точность системных часов. Обычно это не является проблемой, за исключением систем с очень точными временными ограничениями.

Временные диаграммы очень просты. Время откладывается на горизонтальной оси слева направо. Линии жизни и их состояния (или определённые условия, накладываемые на линии жизни) располагаются вертикально. Переходы между состояниями линий жизни и условиями представляются в виде графика.

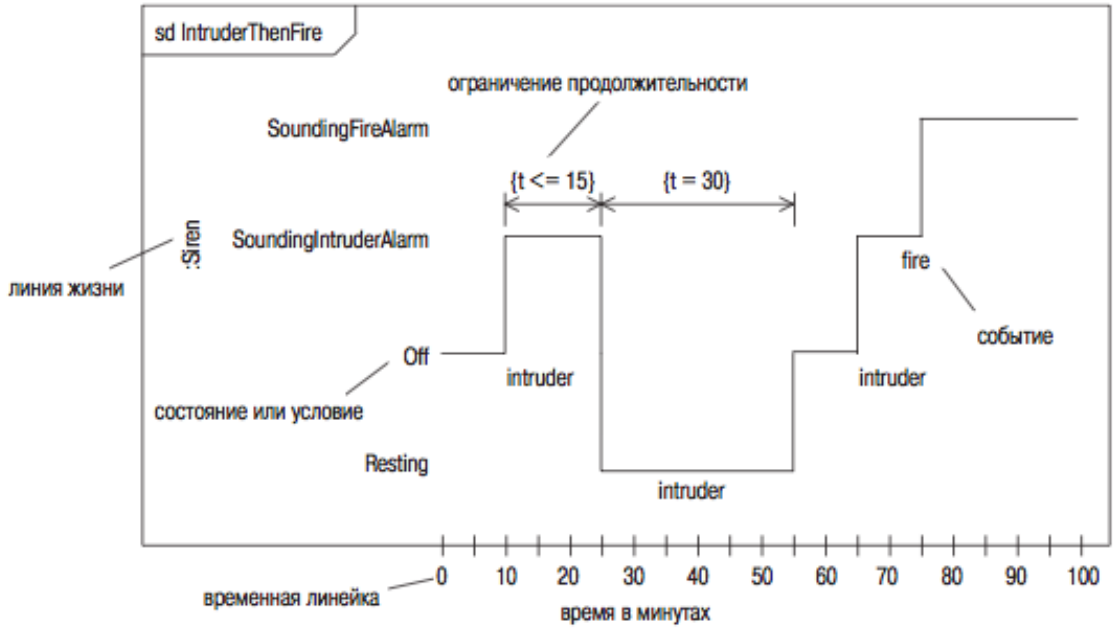


Рис. 8.7. Временная диаграмма

Временные диаграммы также можно представить в более компактной форме, когда состояния располагаются горизонтально.

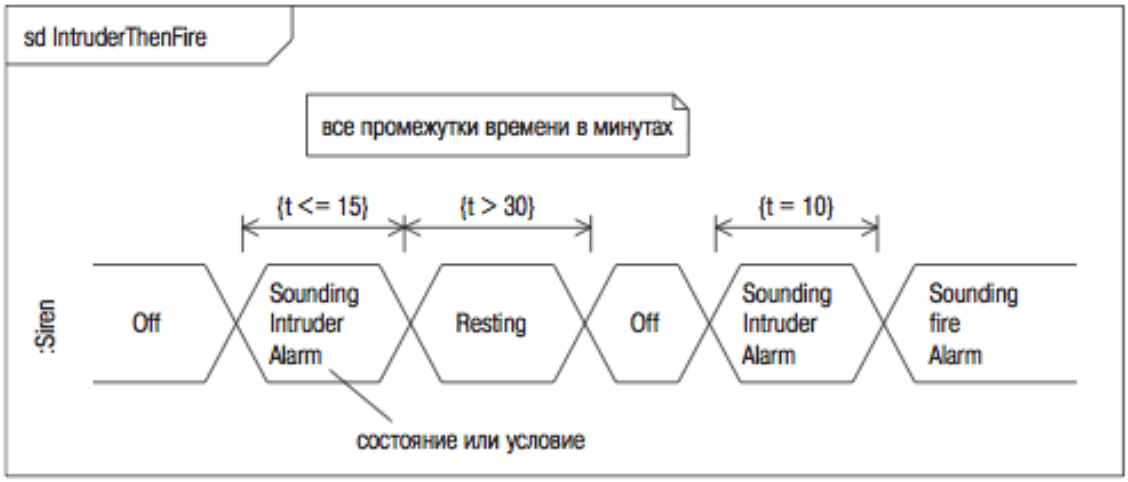


Рис. 7.8. Компактная форма временной диаграммы

8.4.Задание на лабораторную работу.

- Изучить теоретическую часть лабораторной работы.
- Для произвольной программной системы описать динамику поведения программной системы с помощью автоматной схемы.
- Оформить структуру программ в виде диаграммы состояний.
- Оформить отчёт по лабораторной работе