Министерство науки и образования РФ

Государственное бюджетное образовательное учреждения высшего профессионального образования

Московский государственный технический университет имени Н.Э.Баумана

(исследовательский университет)

Кафедра Системы обработки данных и управления

А.В.Балдин

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ**

к выполнению домашнего задания

по дисциплине

**Объектно ориентированное проектирование автоматизированных систем обработки информации и управления**

Специальность: 09.04.01

**ОПИСАНИЕ ПРОЕКТА ИНФОРМАЦИОННОЙ СИСТЕМЫ НА ЯЗЫКЕ UML 2**

Москва 2019

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 4

1 ОБЩИЕ ТРЕБОВАНИЯ К ОПИСАНИЮ ПРОЕКТА ИНФОРМАЦИОННОЙ СИСТЕМЫ 4

2 Домашнее задание № 1. ОПИСАНИЕ статической структуры ПРОЕКТИРУЕМОЙ информационной системы 7

2.1 Цель домашнего задания 7

2.2 Требования к домашнему заданию 7

2.3 Теоретическая часть 8

2.3.1 Контекст системы (границы системы) 8

2.3.2 Диаграмма прецедентов 9

2.3.3 Нотация классов в UML 10

2.3.4 Отношения 11

2.3.5 Возможность навигации   13

2.3.6 Квалифицированные ассоциации 14

2.3.7 Зависимость 15

2.3.8 Обобщение 16

2.3.9 Наследование классов 16

2.3.10 Интерфейсы 17

2.3.11 Порты 17

2.3.12 Компонент 18

2.3.13 Подсистемы 19

2.3.14 Пакеты 20

2.3.15 Вложенные пакеты 21

2.3.16 Зависимости пакетов 22

2.3.17 Диаграмма развёртывания 24

2.4 Вопросы рубежного контроля 27

3 Домашнее задание № 2. ОПИСАНИЕ ДИНАМИКИ ПОВЕДЕНИЯ ПРОЕКТИРУЕМОЙ информационной системы 28

3.1 Цель домашнего задания 28

3.2 Требования к домашнему заданию 28

3.3 Теоретическая часть 28

3.3.1 Диаграммы динамического поведения 28

3.3.2 Диаграммы последовательностей 29

3.3.3 Инварианты состояния и ограничения 31

3.3.4 Комбинированные фрагменты и операторы 33

3.3.5 Ветвление с помощью операторов opt и alt 35

3.3.6 Организация итераций операторами loop и break 37

3.3.7 Коммуникационные диаграммы 40

3.3.7.1 Итерации на коммуникационных диаграммах 41

3.3.7.2 Ветвление на коммуникационных диаграммах 42

3.3.8 Диаграммы деятельности 43

3.3.8.1 Узлы действия 45

3.3.8.2 Узлы управления 45

3.3.8.3 Узлы решения и слияния 46

3.3.8.4 Узлы ветвления и параллелизм 47

3.3.8.5 Разъёмы 48

3.3.8.6 Разделы деятельности 49

3.3.8.7 Прерывание выполнения действий 50

3.3.8.8 Отправка сигналов и приём событий 51

3.3.8.9 Центральный буфер 52

3.3.9 Диаграммы обзора взаимодействий 53

3.3.10 Конечные автоматы 54

3.3.11 Диаграммы состояний 55

3.3.11.1 Синтаксис состояния 55

3.3.11.2 Переходы 57

3.3.11.3 События 57

3.3.11.4 Сигналы 58

3.3.11.5 События изменения 59

3.3.11.6 Ветвление переходов 60

3.3.11.7 Составные состояния 61

3.3.11.8 Ортогональные составные состояния 62

3.3.11.9 Состояния подавтоматов 63

3.3.12 Временные диаграммы 64

3.4 Вопросы рубежного контроля 66

ЛИТЕРАТУРА 67

# ВВЕДЕНИЕ

Домашнее задание по дисциплине «Объектно ориентированное проектирование автоматизированных систем обработки информации и управления» для магистров первого курса обучения по специальности 09.04.01 имеет своей целью получить практические навыки при проектировании информационных систем.

В процессе работы над домашним заданием необходимо представить описание всех аспектов проектирования информационной системы, начиная от формулирования требований и заканчивая развёртыванием информационной системы на технических средствах.

Описание проекта информационной системы основывается на использовании унифицированного языка моделирования UML2. Назначение и возможности этого языка раскрываются в теоретической части дисциплины «Объектно ориентированное проектирование автоматизированных систем обработки информации и управления». При выполнении домашнего задания должны быть на практике применены полученные в теоретическом курсе знания.

Домашнее задание состоит из двух взаимосвязанных частей (двух заданий), которые проектируемую информационную систему рассматривают с точки зрения структуры и с точки зрения динамики поведения. В целом, после выполнения двух домашних заданий должен получиться целостный проект информационной системы.

Сроки выполнения домашних заданий определяются учебным планом по специальности 09.04.01

# ОБЩИЕ ТРЕБОВАНИЯ К ОПИСАНИЮ ПРОЕКТА ИНФОРМАЦИОННОЙ СИСТЕМЫ

Задание на проектирование информационной системы выдаётся каждому студенту индивидуально.

Примерами заданий на проектирование могут быть:

* Информационная система бронирования и продажи билетов в театр
* Информационная система управления грузоперевозками
* Информационная система оформления пропусков на закрытую территорию предприятия
* Информационная система предварительной записи к специалистам в медицинском учреждении
* И другие подобные информационные системы

В процессе выполнения домашнего задания необходимо отразить статические и динамические аспекты проекта информационной системы. Описание проекта должно основываться на диаграммах языка UML2. Проект должен содержать 2-3 (и более) диаграмм каждого вида с использованием комбинированных структур. Каждая диаграмма должна иметь краткое описание используемых элементов с точки зрения предметной области.

Существует тринадцать различных типов UML диаграмм, все они приведены на рис. 1.1.

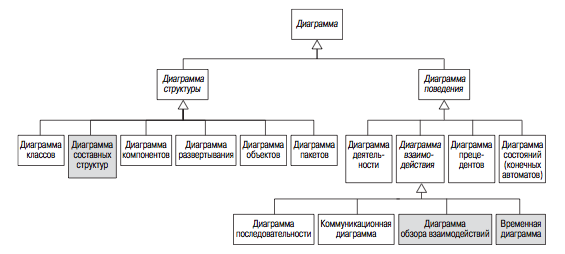


Рис. 1.1. Типы UML диаграмм

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 программной̆ системы.

Домашнее задание №1 должно содержать статические аспекты описания проектируемой системы

Домашнее задание №2 должно содержать динамические аспекты описания проектируемой системы

В целом, задания должны охватить все аспекты описания проектируемой информационной системы.

Отчёт по домашнему заданию должен содержать:

* Название домашнего задание
* Цель работы
* Краткое описание предметной области
* Диаграммы на языке UML, отражающие отдельные аспекты проектируемой информационной системы
* Текстовое пояснение к каждой представленной диаграмме

# Домашнее задание № 1. ОПИСАНИЕ статической структуры ПРОЕКТИРУЕМОЙ информационной системы

## Цель домашнего задания

Целью работы является построение статического описания проекта структуры информационной системы.

## Требования к домашнему заданию

В домашнем задании №1 необходимо:

1. Определить функциональные требования к проектируемой информационной системе. Функциональные требования должны быть представлены в виде нескольких диаграмм прецедентов. Следует иметь ввиду, что прецеденты диаграммы в дальнейшем, во втором домашнем задании, должны быть уточнены в виде описания бизнес процесса в соответствующей динамической диаграмме.
2. Определить структуру программного обеспечения информационной системы (состава программных модулей) на основе диаграммы компонентов.
3. Определить состав классов и их взаимодействие для одного, или нескольких программных компонентов с помощью диаграммы классов.
4. Определить структуру базы данных с помощью диаграммы классов
5. Определить структуру диалогового взаимодействия пользователя с системой с помощью диаграммы классов для одного из компонентов системы
6. Определить структуру технических средств, на которых будет разворачиваться информационная система с помощью диаграммы развёртывания.
7. Определить состав программных комплексов и их установку на технические средства с помощью диаграммы пакетов.

Все диаграммы должны быть снабжены текстовыми пояснениями их назначения и особенностей.

## Теоретическая часть

### Контекст системы (границы системы)

Первое, что необходимо сделать при построении системы, – обозначить её границы. Иначе говоря, надо определить, что является *частью* системы (находится внутри границ системы) и что находится *вне* системы (вне её 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 UML 2 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 данной̆ системой̆. Он может представлять роль пользователя или роль, исполняемую другой̆ системой̆ или частью аппаратных средств, которые касаются границ системы.

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

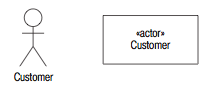


Рис. 2.1.Варианты изображения актера

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 людей̆ или сущностей̆.
* Один человек или сущность может играть по отношению к системе множество ролей̆ одновременно или последовательно во времени. Например, вы составляете и ведете учебные курсы. С точки зрения системы планирования курсов вы играете две роли: Trainer (инструктор) и CourseAuthor (автор курса).
* У каждого актера должно быть короткое, осмысленное с прикладной̆ точки зрения имя.

Каждого актера 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 актера. Это «вариант использования» системы конкретным актером:

* прецеденты *всегда* инициируются актером;
* прецеденты *всегда* описываются с точки зрения актеров.

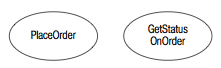


Рис. 2.2.Пиктограмма прецедента

### 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. Актеры 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. 2.3.

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

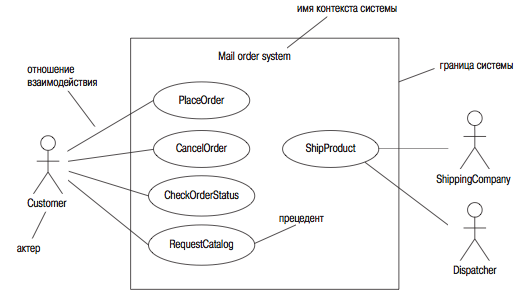


Рис. 2.3**.** Диаграмма прецедентов

### Нотация классов в UML

Визуальный синтаксис UML для класса очень богат. Чтобы синтаксис был управляемым, в UML 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. 2.4. показано все.

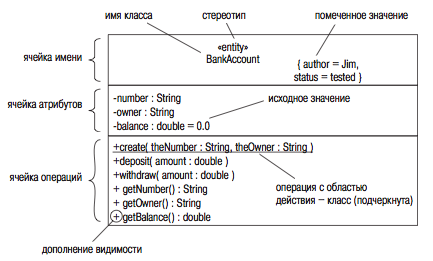


Рис. 2.4. Нотация классов в UML

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).

### Отношения

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

Обычно одна связь соединяет только два объекта. Однако UML 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 UML 2 допускает три разных способа отображения возможности навигации. Самое распространенное обозначение:

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

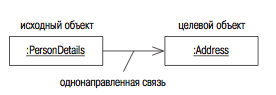
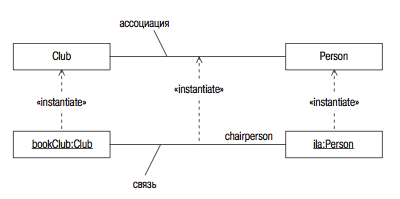


Рис. 2.5. Однонаправленная связь между объектами

**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. 2.6.** Отношения между классами и объектами и между связями и ассоциациями

Ассоциации могут иметь:

* имя ассоциации
* имена ролей • кратность
* • возможность навигации

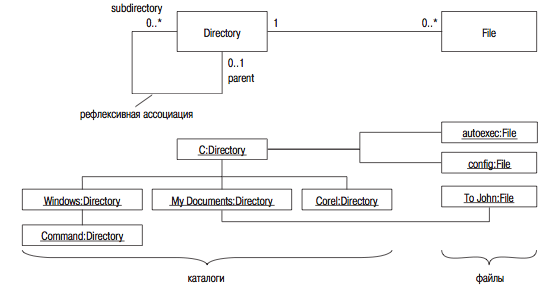


Рис. 2.7. Пример рефлексивной ассоциации: вверху – диаграмма классов, внизу – диаграмма объектов

### 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».

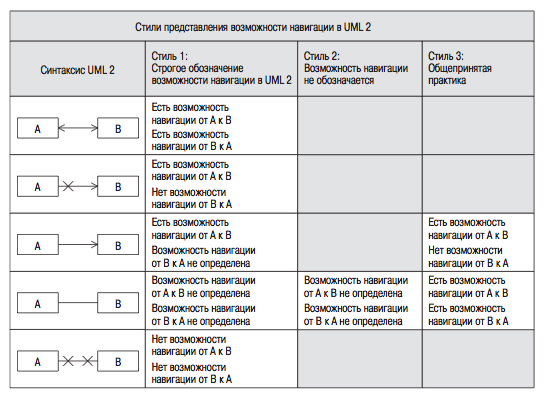


Рис. 2.8. Стили представления возможности навигации в UML 2

### Квалифицированные ассоциации

Квалифицированные ассоциации могут использоваться для превращения ассоциации n ко многим в ассоциацию n к одному путем 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.

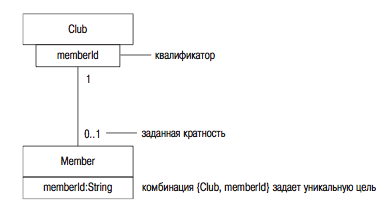


Рис. 2.9. Квалификатор превращает отношение один ко многим в отношение один к одному

### 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.

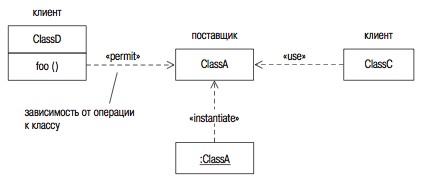


Рис. 2.10. Типы зависимостей

### 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.

### 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.

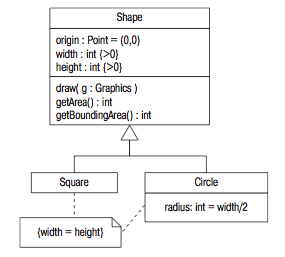


Рис. 2.11. Наследование характеристик суперкласса

### 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.

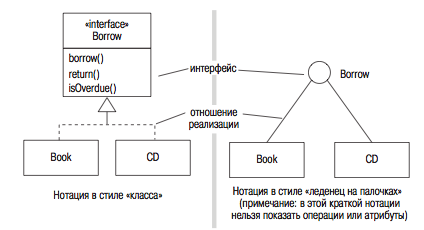


Рис. 2.12. Два варианта синтаксиса для предоставляемых интерфейсов

### 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. 2.13. 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.

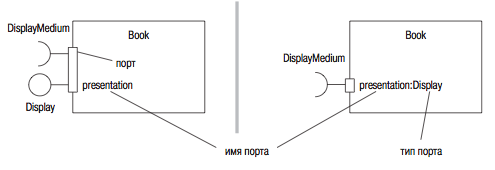


Рис. 2.13. Два варианта нотации порта

### Компонент

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

Компонент как черный ящик, внешнее поведение которого полностью определяется его предоставляемыми и требуемыми интерфейсами. Поэтому один компонент может быть заменен 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 соединенными с ним отношением зависимости. Обе формы синтаксически эквивалентны, хотя первая нотация нам кажется более наглядной.

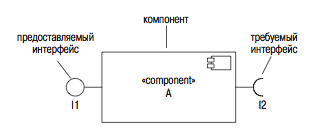


Рис. 2.14. Нотация компонента

Если у компонента есть внутренняя структура, как правило, он будет делегировать обязанности, определенные его интерфейсами, своим внутренним частям. На рис.2.14. 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.

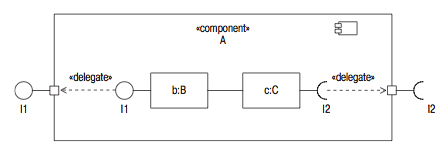


Рис. 2.15. Внутренняя структура компонента

### 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.

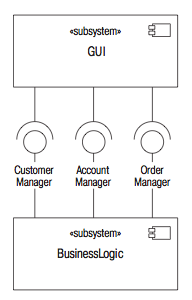


Рис. 2.16. Подсистемы и интерфейсы

### Пакеты

Пакет – это группирующая сущность. Это контейнер и владелец элементов модели. У каждого пакета есть свое пространство имен, 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 имен, 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 имен. Это означает, что пакет создает границу, в рамках которой имена всех элементов должны быть уникальными. Это также означает, что если элементу из одного пространства имен необходимо обратиться к элементу из другого пространства имен, он должен указать и имя необходимого элемента, *и* путь к этому элементу, чтобы его можно было найти в пространствах имен.

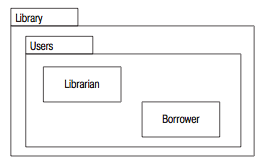


Рис. 2.17. Синтаксис вложения путём встраивания

### 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.

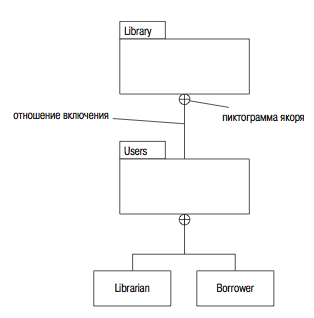


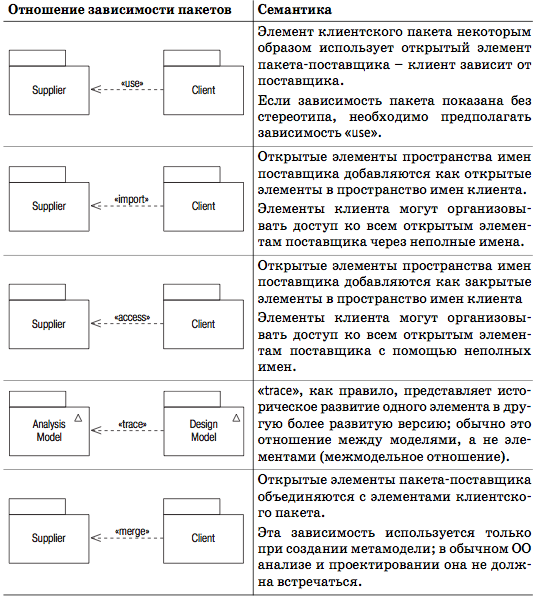
Рис. 2.18.**.** Синтаксис сложного вложения

### 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 2.1.



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

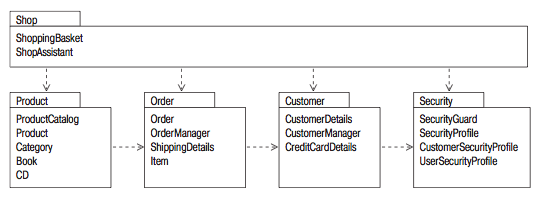


Рис. 2.19. Модель пакетов анализа для системы е-коммерции

### 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 развёртывания.

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 конкретный̆ экземпляр типа программного обеспечения, например определённую копию FrameMaker (www. *adobe.com),* использованную для написания этой книги, или конкретный̆ JAR файл. Если детали конкретных экземпляров неизвестны (или неважны), могут использоваться анонимные экземпляры.

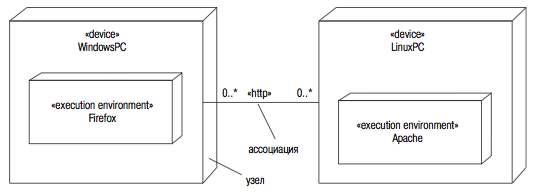


Рис. 2.20. Дескрипторная форма диаграммы развёртывания

Дескрипторная форма диаграмм развёртывания хороша для моделирования типа физической̆ архитектуры, а экземплярная форма – для моделирования фактического развёртывания этой̆ архитектуры на конкретном сайте.

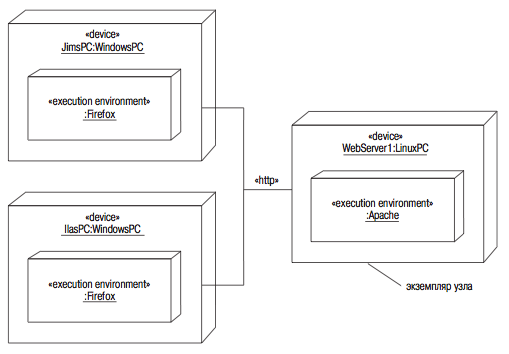


Рис. 2.21.Экземплярная форма диаграммы развёртывания

Для стереотипов можно придумать собственные пиктограммы, напоминающие реальное оборудование, и затем использовать эти символы на диаграмме развёртывания. Такой подход упрощает восприятие диаграммы. Пример полностью визуализированной̆ дескрипторной̆ формы диаграммы развёртывания приведён на рис. 2.22.

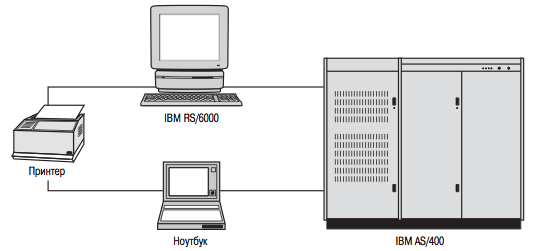


Рис. 2.22.Визуализированная дескрипторная форма диаграммы развёртывания

## Вопросы рубежного контроля

1. Вид и назначение диаграммы прецедентов
2. Актёры и границы информационной системы
3. Возможные связи между прецедентами на диаграмме прецедентов
4. Стереотипы связей на диаграмме прецедентов
5. Вид и назначение диаграммы классов
6. Различные виды изображения классов на диаграмме классов
7. Абстрактные классы и интерфейсы
8. Интерфейсы. Назначение и изображение на диаграмме классов. Предоставление интерфейсов и использование интерфейсов. Реализация интерфейсов
9. Отношения на диаграмме классов. Виды отношений
10. Отношения на диаграмме классов. Стереотипы отношений
11. Отношение ассоциации. Квалифицированная ассоциация
12. Отношение зависимости
13. Отношение обобщения. Различные виды обобщения
14. Отношение наследования. Различные виды наследования
15. Назначение портов на диаграмме классов
16. Класс-ассоциация
17. Моделирование базы данных на основе диаграммы классов
18. Моделирование пользовательского интерфейса на основе диаграммы классов
19. Вид и назначение диаграммы компонентов
20. Компоненты с внутренней структурой
21. Вид и назначение диаграммы пакетов
22. Вложенные пакеты
23. Отношения между пакетами
24. Связь между диаграммой пакетов и диаграммой компонентов
25. Вид и назначение диаграммы развёртывания
26. Связь диаграммы развёртывания и диаграммы пакетов

# Домашнее задание № 2. ОПИСАНИЕ ДИНАМИКИ ПОВЕДЕНИЯ ПРОЕКТИРУЕМОЙ информационной системы

## Цель домашнего задания

Целью работы является описание динамического поведения описания проектируемой информационной системы.

## Требования к домашнему заданию

В домашнем задании №2 необходимо:

1. Определить взаимодействие отдельных компонент информационной системы с помощью диаграммы взаимодействия.
2. Определить последовательность информационного взаимодействия между компонентами информационной системы с помощью диаграммы последовательностей.
3. Определить структуру информационного взаимодействия между компонентами информационной систем на основе коммуникационной диаграммы.
4. Определить бизнес процесс работы системы (части системы) с помощью диаграммы деятельности.
5. Определить бизнес процесс работы системы (части системы) с помощью диаграммы обзора взаимодействия.
6. Определить бизнес процесс реализации одного из прецедентов из домашнего задания №1 с помощью диаграммы конечных автоматов.
7. Определить временную последовательность работы компонент системы на основе временной диаграммы

Все диаграммы должны быть снабжены текстовыми пояснениями их назначения и особенностей

## Теоретическая часть

### Диаграммы динамического поведения

**Диаграммы последовательностей̆ (sequence diagrams)** акцентируют внимание на временноӗ упорядоченности сообщений. Обычно пользователи лучше понимают диаграммы последовательностей̆, 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 UML 2 эти диаграммы предлагают только лишь подмножество функциональности диаграмм последовательностей.

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

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

### Диаграммы последовательностей

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



Рис. 3.1Диаграмма последовательностей

На рис. 3.1. показана диаграмма последовательностей, реализующая поведение информационной системы.

Время на диаграммах последовательностей развивается сверху вниз, линии жизни выполняются слева направо. Местоположение событий 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 *фокусом управления.*

Уничтожение объекта обозначается большим крестом, завершающим линию жизни. Если момент уничтожения объекта неизвестен или неважен, линия жизни завершается без креста.

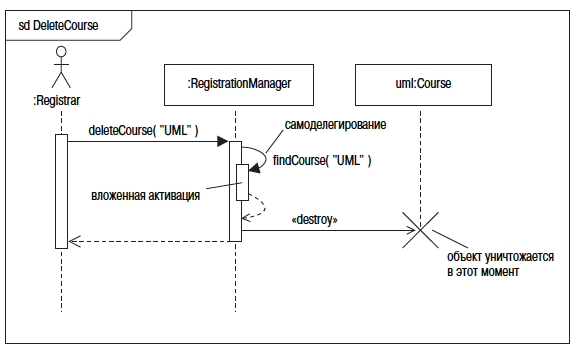


Рис. 3.2Самоделегирование

На рис. 3.2 также представлено самоделегирование: линия жизни объекта посылает сообщение самой себе. Это создает вложенную активацию.

Объекты предлагают ряд открытых сервисов (открытых операций), которые могут быть вызваны клиентскими объектами. Но, как правило, они также имеют и закрытые «вспомогательные» операции, разработанные специально для вызова самим объектом. В данном примере линия жизни :RegistrationManager посылает сама себе сообщение findCourse( "UML" ) , чтобы найти UML\_объект курса, если таковой существует.

Закрытые операции объекта могут быть вызваны только самим объектом средствами самоделегирования.

### Инварианты состояния и ограничения

Сообщение, получаемое экземпляром, может обусловить изменение его состояния.

Состояние экземпляров на линиях жизни объектов можно показать с помощью инвариантов состояния (state invariants ).

Добавление инвариантов состояния на диаграмму последовательностей может быть весьма полезным методом анализа, поскольку позволяет фиксировать ключевые состояния жизненного цикла линии жизни. Они отражают важные состояния системы и могут быть основой для диаграммы автоматов.

На рис. 3.3. показано, что сразу после создания экземпляр Order переходит в состояние unpaid (не оплачено) (изображается в виде прямоугольника со скруглёнными углами). Это сообщает о том, что все вновь создаваемые экземпляры Order находятся в состоянии unpaid.

При получении сообщения acceptPayment() (принять платёж), которое должно обозначать полную оплату, экземпляр переходит в состояние paid (оплачено). После этого экземпляр DeliveryManager (менеджер по доставке) посылает сообщение deliver() (доставить). Он передаёт это сообщение экземпляру Order, что приводит к его переходу в состояние delivered (доставлено).

Рис. 3.3 иллюстрирует применение ограничений. Они записываются в фигурных скобках и размещаются на (или рядом) линиях жизни. Ограничение, обозначенное на линии жизни, указывает на что-то, что должно быть истинным для экземпляров, начиная с этого момента и далее.

Ограничения часто формулируются на естественном языке, хотя в UML есть формальный язык ограничений.

На рисунке показано ограничение продолжительности. Линия жизни объекта :Customer имеет две метки, A и B , и ограничение {B – A <= 28 days} .

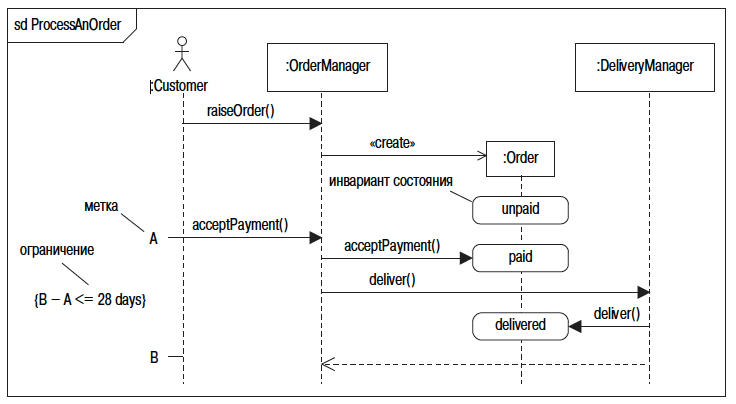


Рис. 3.3 Состояния и ограничения

Это говорит о том, что промежуток времени между точками A и B не должен превышать 28 дней. Точка A обозначает момент оплаты, а точка B – момент доставки товаров покупателю. На естественном языке это ограничение означает, что «заказ должен быть доставлен в течение 28 дней после получения платежа».

На линию жизни может быть помещен любой тип ограничения. Широко распространены ограничения значений атрибутов экземпляров.

### Комбинированные фрагменты и операторы

Диаграммы последовательностей можно разделить на области, называемые комбинированными фрагментами (combined fragments). Рис. 3.4 иллюстрирует синтаксис комбинированного фрагмента.

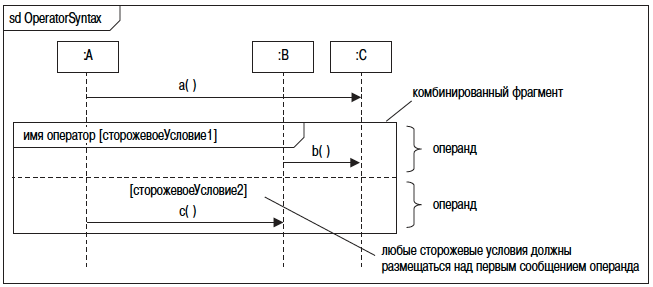


Рис. 3.4. Синтаксис комбинированного фрагмента

Комбинированные фрагменты разделяют диаграмму последовательностей на области с различным поведением.

У каждого комбинированного фрагмента есть один оператор (operator), один или более операндов (operands), и нуль или более сторожевых условий (guard conditions).

Оператор определяет, как исполняются его операнды.

Сторожевые условия определяют, будут ли исполняться эти операнды. Сторожевое условие – это логическое выражение, и операнд исполняется тогда и только тогда, когда это выражение истинно. Одно сторожевое условие может применяться ко всем операндам или каждый операнд может иметь собственное уникальное сторожевое условие.

Сторожевые условия определяют, будут ли выполняться данные операнды.

Полный список операторов приведен в табл. 3.1.

Таблица 3.1

|  |  |  |
| --- | --- | --- |
| **Оператор** | **Полное имя** | **Семантика** |
| opt | option | Единственный операнд выполняется, если условие истинно (как if … then). |
| alt | alternatives | Выполняется тот операнд, условие которого истинно. Вместо логического выражения может использоваться ключевое слово else (как select … case). |
| loop | loop | Имеет специальный синтаксис: loop min, max [condition] повторять минимальное количество раз, за\_тем, пока условие истинно, повторять еще (max – min) число раз. |
| break | break | Если сторожевое условие истинно, выполняется операнд, а не все остальное взаимодействие, в которое он входит. |
| ref | reference | Комбинированный фрагмент ссылается на другое взаимодействие. |
| par | parallel | Все операнды выполняются параллельно. |
| critical | critical | Операнд выполняется автоматически без прерывания. |
| seq | weak sequencing | Все операнды выполняются параллельно при условии выполнения следующего ограничения: последовательность поступления событий на одну линию жизни от разных операндов такая же, как и последовательность операндов. В результате наблюдается слабая форма упорядочения; отсюда название (weak sequencing – слабое упорядочение). |
| strict | strict sequencing | Операнды выполняются в строгой последовательности. |
| neg | negative | Операнд демонстрирует неверные взаимодействия. Применяется, когда необходимо показать, что не должно произойти. |
| ignore | ignore | Перечисляет сообщения, которые намеренно исключены из взаимодействия. Разделенный запятыми список имён проигнорированных сообщений в фигурных скобках помещается после имени оператора, например {m1, m2, m3}. Например, взаимодействие может представлять тестовый пример, в котором принято решение игнорировать некоторые сообщения. |
| consider | consider | Перечисляет сообщения, намеренно включённые во взаимодействие. Разделенный запятыми список имён сообщений в фигурных скобках помещается после имени оператора. Например, взаимодействие может представлять тестовый пример, в который решено включить подмножество возможных сообщений. |
| assert | assertion | Операнд является единственно возможным допустимым поведением в данный момент взаимодействия, любое другое поведение было бы ошибочным. Используется как средство обозначения того, что некоторое поведение должно иметь место в определенной точке взаимодействия. |

Самые важные из них: opt , alt , loop , break , ref , par и critical . Операторы par и critical имеют отношение к параллелизму. Остальные операторы используются редко. В таблице предоставлен достаточный объем информации для их применения в случае необходимости.

### Ветвление с помощью операторов opt и alt

Рисунок 3.5. иллюстрирует синтаксис операторов opt и alt .

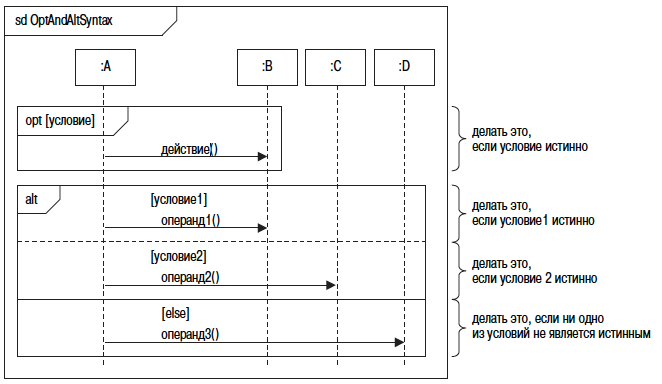


Рис. 3.5. Операторы opt и alt

Оператор opt показывает, что его единственный операнд выполняется, если сторожевое условие истинно. В противном случае выполнение продолжается после комбинированного фрагмента.

opt эквивалентен программной конструкции:

if (условие1) then действие1

alt создает несколько ветвей.

Оператор alt предоставляет выбор между рядом альтернатив. Каждый из его операндов имеет собственное сторожевое условие и будет выполняться, если только условие истинно. Необязательный операнд, заданный сторожевым условием [else ], выполняется, если ни одно из других сторожевых условий не является истинным.

Это означает, что выполняться может только один из операндов alt, т.е. сторожевые условия всех операндов должны быть взаимоисключающими. Истинность более одного из сторожевых условий в любой момент времени является условием возникновения ошибки. Спецификация UML не описывает поведение alt в этом случае. alt эквивалентен программной конструкции:

if (условие1) then

операнд 1

else if (условие2) then

операнд 2

Оператор opt семантически эквивалентен оператору alt с одним операндом.

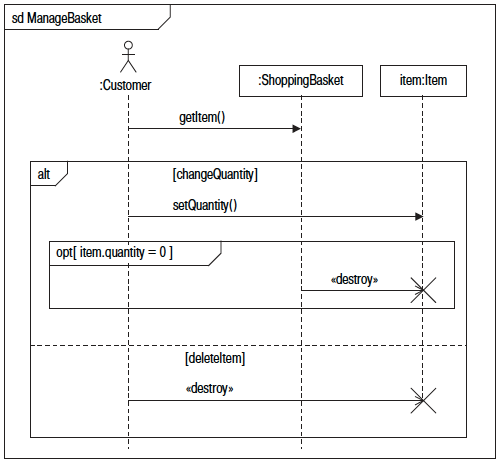


Рис. 3.6. Оператор opt

### Организация итераций операторами loop и break

loop позволяет моделировать итерацию.

Очень часто на диаграммах последовательностей необходимо показать циклы. Сделать это можно с помощью операторов loop и break, как показано на рис. 3.7.

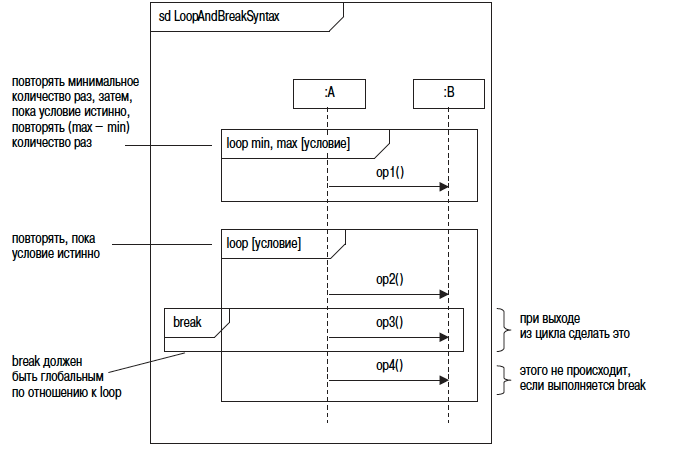


Рис. 3.7. Отображение циклов на диаграмме последовательностей с помощью операторов loop и break

Оператор loop действует следующим образом:

loop min количество раз then

while (условие истинно)

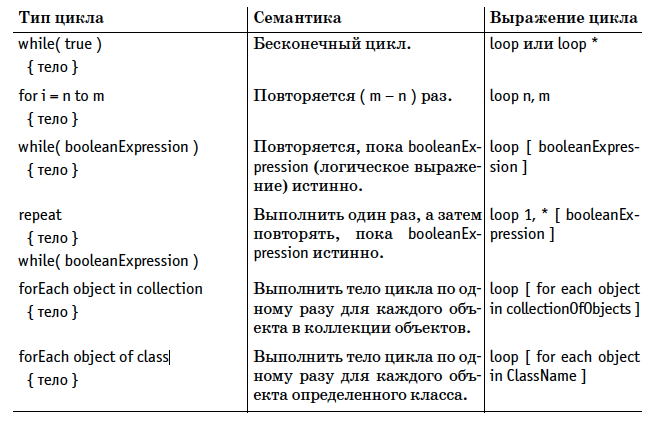
loop (max \_ min) количество раз

В синтаксисе loop необходимо обратить внимание на следующие моменты:

* оператор loop без max, min или condition является бесконечным циклом;
* если задано только min , значит, max = min ;
* condition обычно является логическим выражением, но может быть и произвольным текстом, смысл которого ясен.

loop может может использоваться для поддержки огромного числа разнообразных циклов. Некоторые из самых распространённых циклов перечислены в табл. 3.2.

Таблица 3. 2.



Когда сторожевое условие break становится истинным, выполняется операнд break, цикл прерывается.

Оператор break может использоваться для обозначения условия прекращения loop и того, что происходит после этого. У оператора break только одно сторожевое условие, и если оно истинно, исполняется тело break , а loop прерывается. Самое главное, что после завершения break loop не возобновляется.

Комбинированный фрагмент break логически находится вне цикла: он не является его частью. Поэтому фрагмент break всегда должен изображаться вне loop , но пересекаться с ним, как показано на рис. 3.8.

Самым распространенным применением циклов является перебор коллекции объектов.

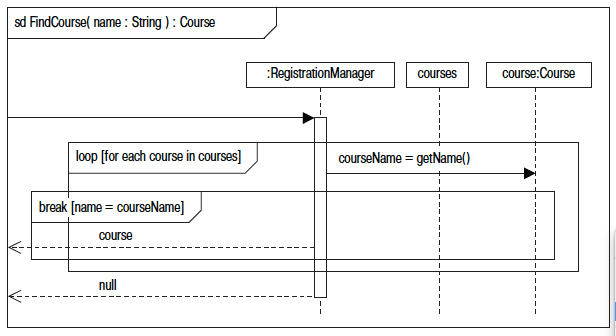


Рис. 3.8. Комбинированный фрагмент break

### Коммуникационные диаграммы

Коммуникационные диаграммы акцентируют внимание на структурных аспектах взаимодействия, на том, как соединяются линии жизни. В UML 2 их семантика довольно слаба по сравнению с диаграммами последовательностей.

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 путём иерархическоӗ нумерации.

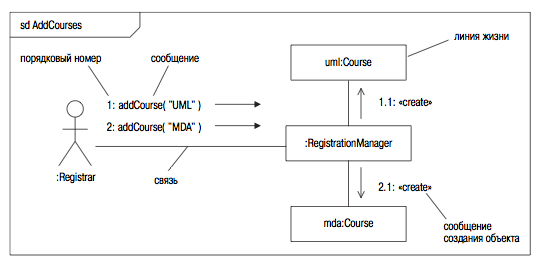


Рис. 3.9. Коммуникационная диаграмма прецедента AddCourses

Главное необходимо понимать, что в результате отправления сообщения вызывается некоторая операция экземпляра и что сложная нумерация сообщений указывает на вложенность вызовов операций.

#### Итерации на коммуникационных диаграммах

Показать итерацию на коммуникационных диаграммах можно с помощью выражения, описывающего итерацию. Оно включает спецификатор итерации (\* ) и (необязательный) блок итерации.

UML 2 не определяет никакого конкретного синтаксиса для блоков итераций, поэтому может использоваться все, что имеет смысл. Обычно хорошим вариантом являются код или псевдокод.

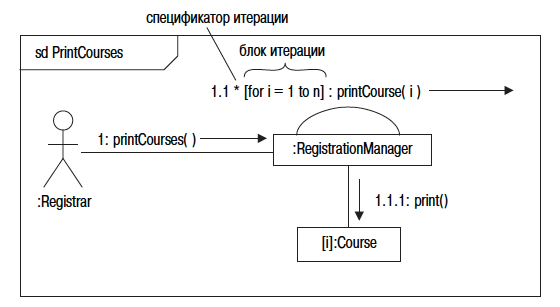


Рис. 3.10. Итерации на коммуникационной диаграмме

#### Ветвление на коммуникационных диаграммах

Ветвление можно смоделировать, добавив в сообщения сторожевые условия. Такие сообщения посылаются только тогда, когда сторожевое условие становится истинным.

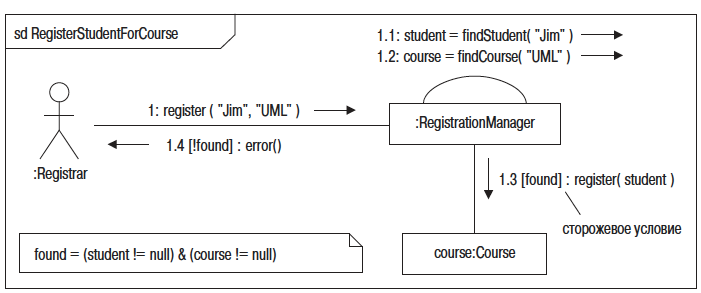


Рис. 3.11. Ветвление на коммуникационной диаграмме

Условия не имеют формального синтаксиса, но обычно являются выражениями, в которых используются временные переменные области действия текущего фокуса управления или атрибуты классов, участвующих во взаимодействии.

На коммуникационных диаграммах довольно сложно отчётливо показать ветвления. Диаграмма очень быстро становится достаточно сложной. Поэтому следует показывать на этих диаграммах только очень простое ветвление. Для представления сложных ветвлений больше подходят диаграммы последовательностей.

### Диаграммы деятельности

В UML 1 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 UML 2 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 UML 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.

Деятельности обычно добавляются к:

• прецедентам;

• классам;

• интерфейсам;

• компонентам;

• кооперациям;

• операциям.

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.

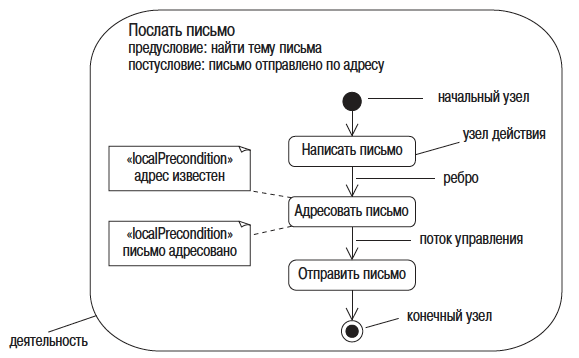
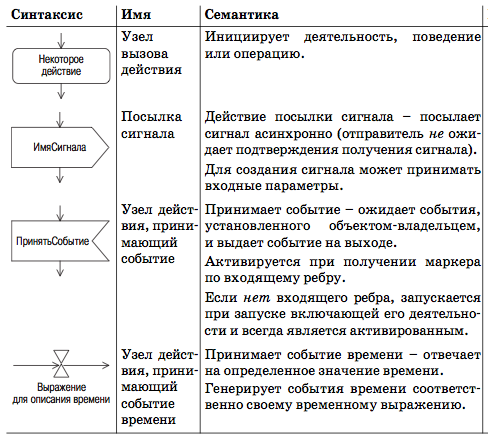


Рис. 3.12. Пример диаграммы деятельности

#### Узлы действия

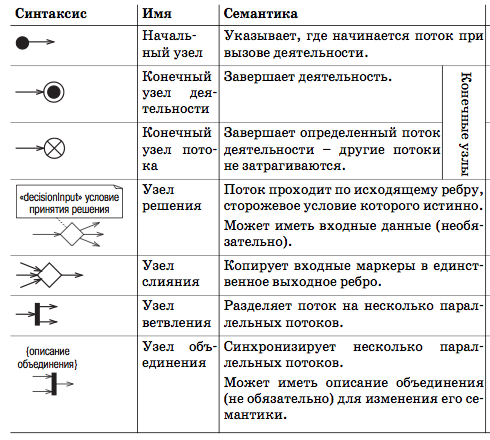
Существует четыре типа узлов действия: они перечислены в табл. 3.3.

Таблица 3.3.



#### Узлы управления

Узлы управления контролируют поток управления деятельности.

Таблица 3.4.

#### Узлы решения и слияния

Узел решения имеет одно входящее ребро и два и более альтернативных исходящих ребер. Маркер, поступающий по входящему ребру, будет предложен всем исходящим ребрам, но пройдет только по одному из них. Узел решения – это перекресток потоков, на котором маркер должен выбрать только один путь.

В узлах слияния сходятся два или более входящих ребра и выходит одно исходящее. Они объединяют все входящие потоки в один исходящий. Семантика слияния очень проста: все маркеры, предлагаемые на входящих ребрах, предлагаются на исходящем ребре. Маркеры и поток не изменяются.

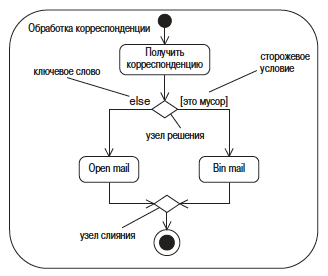


Рис. 3.13. Узлы решения и слияния

Узел слияния и непосредственно следующий за ним узел решения могут быть объединены в один символ,

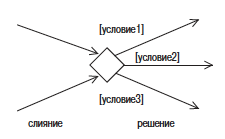


Рис. 3.14. Объединение узла слияния и решения

#### Узлы ветвления и параллелизм

Узел ветвления имеет одно входящее и два или более исходящих ребер. Маркеры, поступающие по входящим рёбрам, дублируются и предлагаются на всех исходящих рёбрах одновременно. Тем самым единственный входящий поток разделяется на несколько параллельных исходящих потоков. У каждого исходящего ребра может быть сторожевое условие, и маркер, как и в узлах решения, может передаваться по исходящему ребру только в случае выполнения сторожевого условия.

Узел объединения синхронизирует и объединяет несколько входящих потоков в единственный исходящий.

В узле объединения несколько входящих рёбер встречаются и объединяются в одно исходящее. Эти узлы синхронизируют потоки: маркер на их единственном исходящем ребре предлагается только после того, как поступили маркеры всех входящих потоков. Они осуществляют операцию логического И над всеми своими входящими рёбрами.

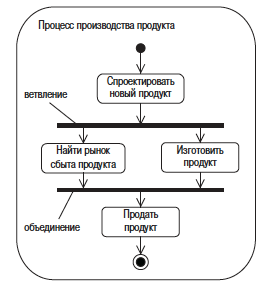


Рис. 3.15. Узлы ветвления и объединения

#### Разъёмы

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

Синтаксис разъема представлен на рис. 3.16. В заданной деятельности каждому исходящему разъему *должен* соответствовать единственный входящий разъем с такой же меткой. Метки – это идентификаторы разъема, никакой другой семантики у них нет. Обычно в их качестве выступают буквы алфавита.

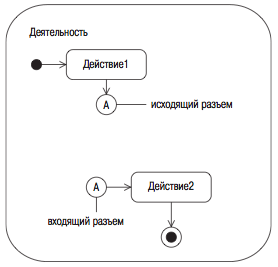


Рис. 3.16. Синтаксис разъёма

#### Разделы деятельности

Чтобы облегчить чтение диаграмм деятельности, можно разбить деятельности на разделы с помощью вертикальных, горизонтальных или кривых линий. Каждый раздел деятельности – это группа взаимосвязанных действий с высоким уровнем вложенности.

Иногда разделы деятельности называют плавательными дорожками (swimlanes). Разбиение на разделы – мощный метод. При правильном использовании он может существенно упростить понимание диаграмм деятельности.

Разделы не имеют собственной семантики, поэтому могут использоваться для разбиения диаграмм деятельности любым удобным способом.

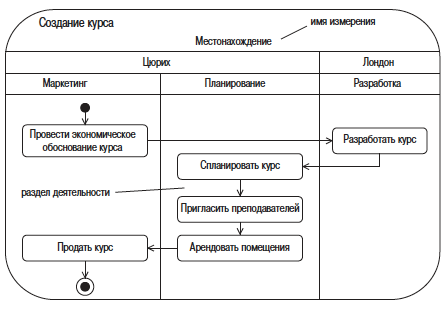


Рис. 3.17. Разделы деятельности

Часто разделы деятельности и параллельные потоки управления взаимосвязаны. Например, обычно разные подразделения или организационные единицы работают параллельно, затем в некоторый момент происходит их синхронизация

#### Прерывание выполнения 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.

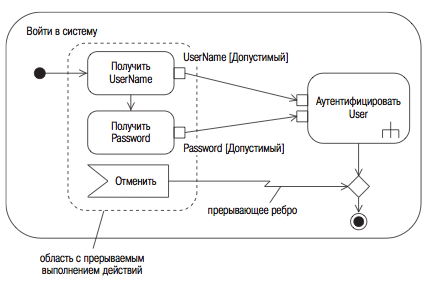


Рис. 3.18. Деятельность с прерываемым выполнением действий

#### 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 «заказ получен»), а при проектировании они могут иллюстрировать асинхронный обмен информацией между разными системами, подсистемами или частями оборудования.

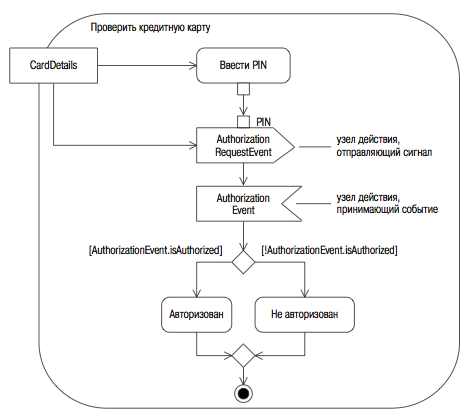


Рис. 3.19. Деятельность Проверить кредитную карту

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

#### Центральный буфер

Центральный буфер – это объектный узел, используемый исключительно как буфер между входными и выходными потоками объектов. Он позволяет объединять несколько входных потоков объектов и распределять объекты между несколькими выходными потоками объектов. Когда объектный узел используется как центральный буфер, он должен быть обозначен стереотипом «centralBuffer».

На рис. 3.20. показан пример, в котором центральный буфер используется для накопления различных типов объектов Order, непрерывно поступающих по нескольким каналам продаж. Объекты Order находятся в буфере в ожидании принимающего их действия Обработать Order. В центральном буфере Order могут сохраняться объекты типа Order или его подклассов: WebOrder (веб заказ), PhoneOrder (заказ по телефону) и PostOrder (заказ по почте).

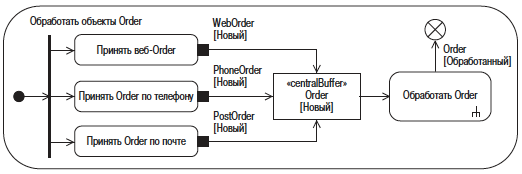


Рис. 3.20. Центральный буфер для хранения объектов

### 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.

Линии жизни, принимающие участие во взаимодействии, могут быть перечислены после ключевого слова lifelines (линии жизни) в заголовке диаграммы.

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.

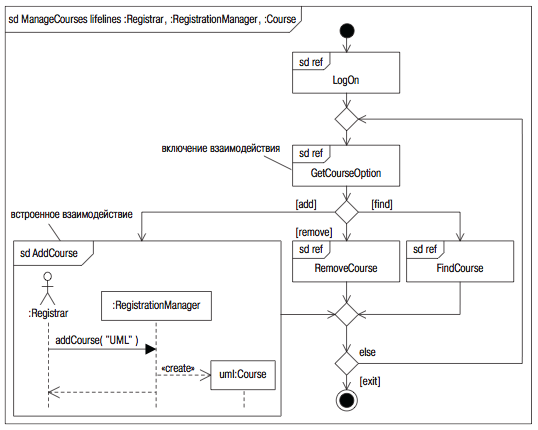


Рис. 3.21.Диаграмма обзора взаимодействий

Диаграммы обзора взаимодействий дают визуальное представление ветвления, параллелизма и итерации, что намного понятней и проще для восприятия. В то же время они переносят 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.

### Конечные автоматы

Конечный автомат моделирует динамическое поведение реактивного объекта.

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.

### Диаграммы состояний

У каждого конечного автомата должно быть начальное состояние (закрашенный кружок), обозначающее первое состояние последовательности. Если смена состояний не бесконечна, должно присутствовать и конечное состояние (светлый кружок), 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.

Диаграмма состояний (state machine diagrams) может изображаться в явно обозначенной рамке, как показано на рис. 5.1, или существовать в неявных рамках, предоставляемых средством моделирования.

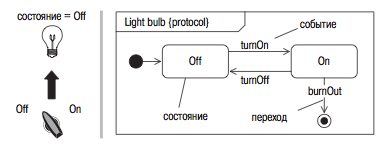


Рис. 3.22.Диаграмма состояний электрической̆ лампочки

#### Синтаксис состояния

UML синтаксис состояния представлен на рис. 3.23.

Каждое состояние поведенческого автомата может содержать нуль или более действий и 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. 3.23. нажатие одной из клавиш клавиатуры, конечно, является заслуживающим внимания событием, но оно не приводит к переходу из состояния EnteringPassword (ввод пароля). Это моделируется как внутреннее событие keypress (нажатие клавиши), которое обусловливает внутренний переход, инициирующий действие отобразить “\*“.

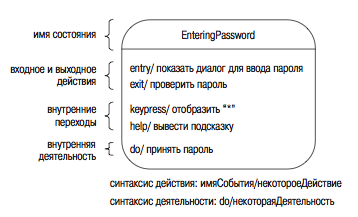


Рис. 3.23.Синтаксис состояния

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.

#### Переходы

UML синтаксис переходов для поведенческих автоматов представлен на рис. 3.24.

Переходы показывают движение между состояниями.

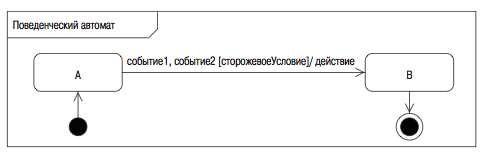


Рис. 3.24.Синтаксис переходов для поведенческих автоматов

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.

#### События

События инициируют переходы. UML 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.

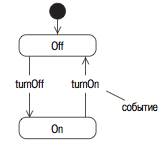


Рис. 3.25.События указаны вне состояний на переходах

Существует четыре семантически различных типа событий:

* событие вызова;
* сигнал;
* событие изменения;
* событие времени.

#### Сигналы

Сигнал – это пакет информации, асинхронно передаваемый между объектами. У сигнала обычно нет операций, потому что онпредназначен только для передачи информации.

На рис. 3.26 представлен конечный автомат для SimpleBankAccount . Теперь при отклонении запроса на снятие денег со счета посылается сигнал. Передача сигнала обозначается выпуклым пятиугольником, внутри которого указывается имя сигнала. Такой же синтаксис используется на диаграммах деятельностей.

Прием сигнала обозначается вогнутым пятиугольником. В нем указывается операция контекстного класса, которая принимает сигнал как параметр.

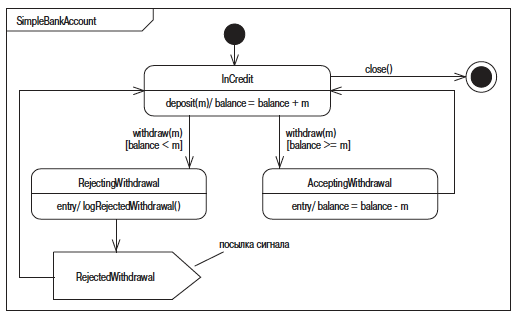


Рис. 3.26.Посылка сигнала

#### События изменения

Событие изменения задаётся в виде логического выражения. Ассоциированное с событием действие выполняется, когда логическое выражение меняет значение с false на true . Все значения логического выражения должны быть константами, глобальными переменными или атрибутами и операциями контекстного класса. С точки зрения реализации событие изменения означает постоянную проверку логического условия при нахождении в определённом состоянии.

На рис. 3.27. конечный автомат SimpleBankAccount был изменен таким образом, что менеджер получает уведомление, если баланс счета становится равным или большим 5000. Это уведомление необходимо для того, чтобы менеджер мог предупредить клиента о других вариантах вклада.

Чтобы событие изменения было инициировано, логическое выражение должно вернуться к значению false , а затем снова перейти к true .

Срабатывание по положительному фронту – это именно то поведение, которое требуется для класса SimpleBankAccount . Действие notifyManager() (уведомить менеджера) будет инициироваться, только когда баланс счета достигнет 5000 или превысит эту сумму. Понятно, что если баланс будет колебаться вокруг 5000, менеджер получит множество уведомлений.

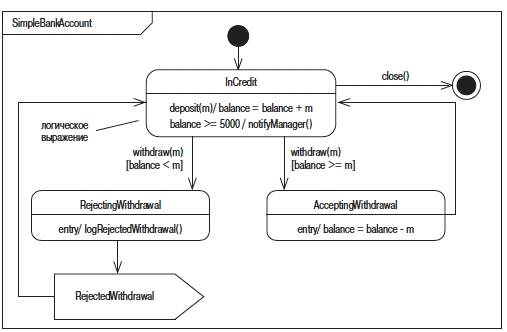


Рис. 3.27. События изменений

#### Ветвление переходов

Псевдосостояние выбора позволяет направлять поток конечного автомата согласно условиям, заданным для исходящих переходов. Например, на рис. 3.28. показан поведенческий автомат простого класса Bank Loan (банковская ссуда). При получении события acceptPayment (принят платеж) объект BankLoan переходит из состояния Unpaid (не оплачен) в одно из трех состояний – FullyPaid (полностью оплачен), OverPaid (переплачен) или PartiallyPaid (частично оплачен) – в зависимости от соотношения объёма платежа (payment) и неоплаченного баланса BankLoan .

Условия, налагаемые на исходящие переходы псевдосостояния выбора, должны быть взаимоисключающими, чтобы гарантировать в любой момент времени срабатывание только одного из них

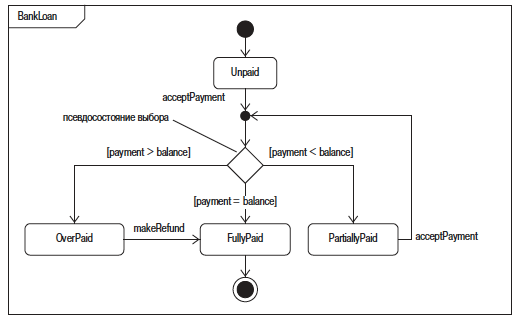


Рис. 3.28.Ветвление

#### 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.

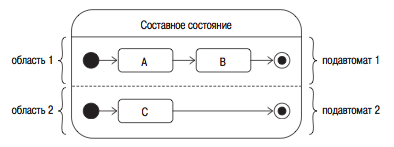


Рис. 3.29.Каждому подавтомату отведена своя область

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

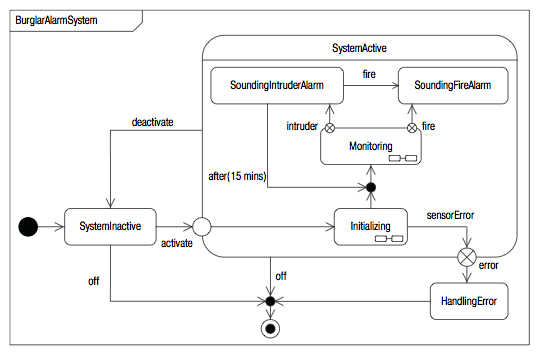


Рис. 3.30.Автомат системы сигнализации

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.

#### Ортогональные составные состояния

Ортогональные составные состояния содержат два или более выполняющихся параллельно подавтоматов.

При входе в составное состояние начинается одновременное выполнение всех его подавтоматов – это неявное ветвление.

Иногда требуется запустить параллельные потоки управления, которые не надо синхронизировать с помощью объединения по их завершении. Этот вариант представлен составным состоянием Monitoring , показанным на диаграмме состояний на рис. 3.31.

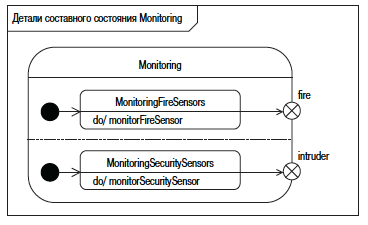


Рис. 3.31. Ортогональные составные состояния

#### Состояния подавтоматов

Состояние подавтомата – это особое состояние, ссылающееся на конечный автомат, представленный на отдельной диаграмме. Это похоже на вызов конечным автоматом подпрограммы другого конечного автомата. Состояния подавтоматов семантически эквивалентны составным состояниям.

Состояния подавтоматов могут использоваться для упрощения сложных автоматов. Конечные автоматы разделяются на отдельные диаграммы, и затем основная диаграмма ссылается на них с помощью состояний подавтоматов.

Состояния подавтоматов могут предоставить способ повторного использования поведения. Поведение описывается на одной диаграмме, и затем при необходимости просто делается ссылка на эту диаграмму.

Состояния подавтоматов именуются следующим образом:

имя состояния : имя используемой диаграммы состояний

На рис. 3.32. представлена диаграмма состояний, описывающая поведение, которое используется в другой диаграмме. Входное и выходное псевдосостояния можно показать на рамке.

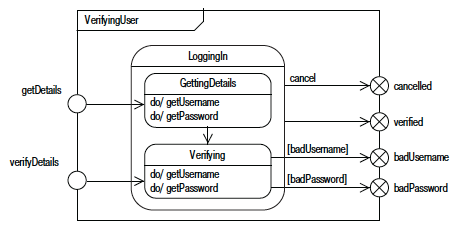


Рис. 3.32. Состояния подавтоматов

### Временные диаграммы

Временные диаграммы очень просты. Время откладывается на горизонтальной̆ 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.

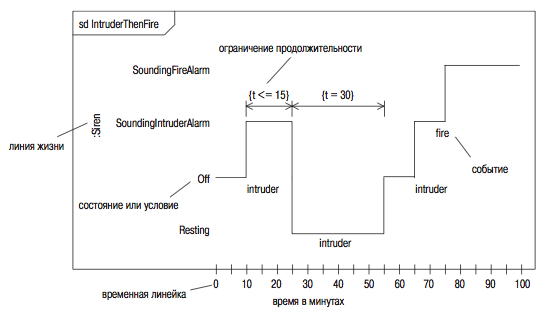


Рис. 3.33.Временная диаграмма

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

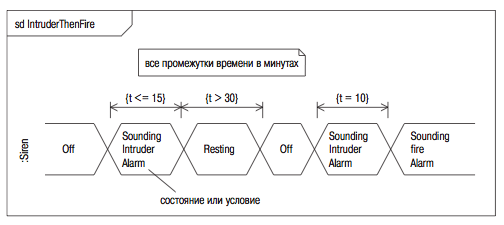


Рис. 3.34.Компактная форма временной диаграммы

## Вопросы рубежного контроля

1. Диаграмма последовательностей. Назначение. Изображение.
2. Инварианты состояния и ограничения
3. Комбинированные фрагменты и операторы
4. Ветвление с помощью операторов opt и alt
5. Организация итераций операторами loop и break
6. Коммуникационные диаграммы
7. Итерации на коммуникационных диаграммах
8. Ветвление на коммуникационных диаграммах
9. Диаграммы деятельности. Назначение. Вид диаграммы
10. Диаграммы деятельности Узлы действия
11. Диаграммы деятельности Узлы управления
12. Диаграммы деятельности Узлы решения и слияния
13. Диаграммы деятельности Узлы ветвления и параллелизм
14. Диаграммы деятельности Разъёмы
15. Диаграммы деятельности Разделы деятельности
16. Диаграммы деятельности Прерывание выполнения действий
17. Диаграммы деятельности Отправка сигналов и приём событий
18. Диаграммы деятельности Центральный буфер
19. Диаграммы обзора взаимодействий Назначение. Вид диаграммы
20. Диаграммы состояний Назначение. Вид диаграммы
21. Диаграммы состояний Синтаксис состояния
22. Диаграммы состояний Переходы
23. Диаграммы состояний События
24. Диаграммы состояний Сигналы
25. Диаграммы состояний События изменения
26. Диаграммы состояний Ветвление переходов
27. Диаграммы состояний Составные состояния
28. Диаграммы состояний Ортогональные составные состояния
29. Диаграммы состояний Состояния подавтоматов
30. Временные диаграммы
31. Компактная форма временной диаграммы

# ЛИТЕРАТУРА

1. Леоненков А.В. Самоучитель UML2 – СПб.: БХВ-Петербург. 2007.- 576 с.
2. Арлоу Д., Нейштадт А. UML2 и унифицированный процесс. 2-е издание. - Пер с англ. - СПб.:Символ Плюс. 2008. -624 с.