Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки кадров

Кафедра «Информатика»

Т. А. Трохова

ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ В UML

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по курсу «Технологии проектирования программного обеспечения информационных систем»
для слушателей специальности 1-40 01 73
«Программное обеспечение информационных систем»

Гомель 2012

УДК 004.4'22(075.8)
ББК 32.973.26-018.2я73
Т76

Рекомендовано кафедрой «Информатика»
ГГТУ им. П. О. Сухого
(*протокол № 12 от 06.06.2012 г.*)

Рецензент: доц. каф. «Инфорационные технологии» ГГТУ им. П. О. Сухого
канд. физ.-мат. наук, доц. *О. А. Кравченко*

Трохова, Т. А.

Т76 Функциональное моделирование программных систем : метод. указания по курсу «Технологии проектирования программного обеспечения информационных систем» для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем» / Т. А. Трохова. – Гомель : ГГТУ им. П. О. Сухого, 2012. – 34 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://alis.gstu.by/StartEK/. – Загл. с титул. экрана.

Рассмотрены основные возможности и структура языка функционального моделирования программных систем UML. Приведено описание канонических диаграмм, а также описание подхода к разработке программной системы с их применением. Даны методические рекомендации к выполнению практических заданий по каждой теме курса.

Для слушателей ИПК и ПК.

УДК 004.4'22(075.8)
ББК 32.973.26-018.2я73

РАЗДЕЛ 1. ОБЩАЯ ХАРАКТЕРИСТИКА ЯЗЫКА UML

1.1. История создания UML

Наиболее популярным языком моделирования в области разработки программного обеспечения на данный момент является графический язык **UML (*Unified Modeling Language*** – унифицированный язык моделировании) – язык, предназначенный для визуализации, специфицирования, конструирования и документирования программных систем.

Выразительных средств этого языка в совокупности с мощными механизмами расширения достаточно для того, чтобы описать любую программную систему со всех точек зрения, актуальных на различных этапах жизненного цикла.

Отдельные языки объектно-ориентированного моделирования стали появляться в период между серединой 1970-х и концом 1980-х годов, когда различные исследователи и программисты предлагали свои подходы к ООАП. В период между 1989–1994 гг. общее число наиболее известных языков моделирования возросло с 10 до более чем 50.

Многие пользователи испытывали серьезные затруднения при выборе языка объектно-ориентированного анализа и проектирования (ООАП), поскольку ни один из них не удовлетворял всем требованиям, предъявляемым к построению моделей сложных систем.

Принятие отдельных методик и графических нотаций в качестве стандартов (IDEF0, IDEF1X) не смогло изменить сложившуюся ситуацию непримиримой конкуренции между ними в начале 90-х годов, которая тоже получила название "войны методов".

К середине 1990-х некоторые из методов были существенно улучшены и приобрели самостоятельное значение при решении различных задач ООАП. Наиболее известными в этот период становятся:

– метод Гради Буча (Grady Booch), получивший условное название Booch или Booch'91, Booch Lite (позже – Booch'93);

– метод Джеймса Румбаха (James Rumbaugh), получивший название Object Modeling Technique – OMT (позже – OMT-2);

– метод Айвара Джекобсона (Ivar Jacobson), получивший название Object-Oriented Software Engineering – OOSE.

История развития языка UML берет начало с октября 1994 года, когда Гради Буч и Джеймс Румбах из Rational Software Corporation начали работу по унификации методов Booch и OMT.

Хотя сами по себе эти методы были достаточно популярны, совместная работа была направлена на изучение всех известных объектно-ориентированных методов с целью объединения их достоинств. При этом Г. Буч и Дж. Румбах сосредоточили усилия на полной унификации результатов своей работы.

Проект так называемого унифицированного метода (Unified Method) версии 0.8 был подготовлен и опубликован в октябре 1995 года. Осенью того же года к ним присоединился А. Джекобсон, главный технолог из компании Objectory AB (Швеция), с целью интеграции своего метода OOSE с двумя предыдущими.

Начиная работу по унификации своих методов, Г. Буч, Дж. Румбах и А. Джекобсон сформулировали следующие требования к языку моделирования. Он должен:

– позволять моделировать не только программное обеспечение, но и более широкие классы систем и бизнес-приложений, с использованием объектно-ориентированных понятий;

– явным образом обеспечивать взаимосвязь между базовыми понятиями для моделей концептуального и физического уровней;

– обеспечивать масштабируемость моделей, что является важной особенностью сложных многоцелевых систем;

– быть понятен аналитикам и программистам, а также должен поддерживаться специальными инструментальными средствами, реализованными на различных компьютерных платформах.

В этот период поддержка разработки языка UML становится одной из целей консорциума OMG (Object Management Group). Именно в рамках OMG создается команда разработчиков под руководством Ричарда Соли, которая будет обеспечивать дальнейшую работу по унификации и стандартизации языка UML.

Компания Rational Software вместе с несколькими организациями, изъявившими желание выделить ресурсы для разработки строгого определения версии 1.0 языка UML, учредила консорциум партнеров UML, в который первоначально вошли такие компании, как Digital Equipment Corp., HP, i-Logix, Intellicorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle, Rational Software, TI и Unisys. Из более чем 800 компаний и организаций, входящих в настоящее время в состав консорциума OMG, особую роль продолжает играть Rational Software Corporation, которая стояла у истоков разработки языка UML. Эта компания разработала и выпустила в продажу одно из первых инструментальных CASE-средств Rational Rose 98, в котором был реализован язык UML.

1.2. Задачи и структура UML

Язык UML предназначен для решения следующих задач.

Первая задача – предоставить в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, специально предназначенный для разработки и документирования моделей сложных систем самого различного целевого назначения.

В настоящее время имеет место некоторый концептуальный разрыв между общей методологией моделирования сложных систем и конкретными инструментальными средствами быстрой разработки приложений. Именно этот разрыв и призван заполнить язык UML.

Для адекватного понимания базовых конструкций языка UML важно не только владеть некоторыми навыками объектно-ориентированного программирования, но и хорошо представлять себе общую проблематику процесса разработки моделей систем.

Вторая задача – снабдить исходные понятия языка UML возможностью расширения и специализации для более точного представления моделей систем в конкретной предметной области.

Третья задача сводится к тому, что описание языка UML должно поддерживать такую спецификацию моделей, которая не зависит от конкретных языков программирования и инструментальных средств проектирования программных систем.

Ни одна из конструкций языка UML не должна зависеть от особенностей ее реализации в известных языках программирования. Но язык UML должен обладать потенциальной возможностью реализации своих конструкций на том или ином языке программирования, в первую очередь имеются в виду языки, поддерживающие концепцию ООАП.

С самой общей точки зрения описание языка UML состоит из двух взаимодействующих частей.

1) Семантика языка UML. Представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на языке UML;

2) Нотация языка UML. Представляет собой графическую нотацию для визуального представления семантики языка UML.

Основным компонентом языка UML является диаграмма.

Диаграмма (diagram) — графическое представление совокупности элементов модели в форме связного графа, вершинам и ребрам (дугам) которого приписывается определенная семантика.

В нотации языка UML определены следующие виды канонических диаграмм:

– вариантов использования (use case diagram);
– классов (class diagram);
– кооперации (collaboration diagram);
– последовательности (sequence diagram);
– состояний (statechart diagram);
– деятельности (activity diagram);
– компонентов (component diagram);
– развертывания (deployment diagram).

Диаграммы представляют статическую структуру приложения (диаграммы вариантов использования, классов и др.), поведенческие аспекты разрабатываемой программной системы (диаграмма деятельности), физические аспекты функционирования системы (диаграммы реализации).

Еще одним компонентом языка является сущность. К сущностям UML относятся структурные сущности (классы, интерфейсы, прецеденты), поведенческие сущности (автоматы), групповые сущности (пакеты), аннатационные сущности (примечания).

К основным элементам языка принадлежат также такие элементы как отношения. Отношения можно классифицировать на отношения ассоциации, зависимостей, обобщения и др.

В методических рекомендациях подробно рассматриваются приемы построения функциональных моделей программных систем в нотациях языка UML.

РАЗДЕЛ 2. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

Тема 1. Диаграмма вариантов использования (диаграмма прецедентов)

Краткие теоретические сведения

Диаграмма вариантов использования (**use case diagram)** описывает функциональное назначение системы – то, что система должна делать в процессе своего функционирования.

Создание диаграммы вариантов использования имеет следующие цели:

– определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;

– сформулировать общие требования к функциональному поведению проектируемой системы;

– разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;

– подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Базовыми элементами диаграммы вариантов использования являются **вариант использования (прецедент) и актер**.

Вариант использования (use case) – внешняя спецификация последовательности действий, которые система или другая сущность могут выполнять в процессе взаимодействия с актерами. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой и собственно выполнение вариантов использования.

Отдельный вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его краткое имя в форме существительного или глагола с пояснительными словами.

Сам текст имени варианта использования должен начинаться с заглавной буквы.

Имя (name) – строка текста, которая используется для идентификации любого элемента модели (см. рис. 1.1).

Рис. 1.1. Графическое обозначение вариантов использования

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач.

Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования.

Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается имя актера (см. рис. 1.2).

Рис. 1.2. Графическое обозначение актера

Имя актера должно быть достаточно информативным с точки зрения семантики. Для этой цели подходят наименования должностей в компании (например, продавец, кассир, менеджер, студент). Не рекомендуется давать актерам имена собственные или названия моделей конкретных устройств, даже если это с очевидностью следует из контекста проекта.

Актеры взаимодействуют с системой посредством передачи и приема сообщений от вариантов использования. Сообщение представляет собой запрос актером сервиса от системы и получение этого сервиса. Это взаимодействие может быть выражено посредством ассоциаций между отдельными актерами и вариантами использования.

Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами.

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

Один актер может взаимодействовать с несколькими вариантами использования. В этом случае этот актер обращается к нескольким сервисам данной системы. В свою очередь один вариант использования может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис.

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

– ассоциации (association relationship);
– включения (include relationship);
– расширения (extend relationship);
– обобщения (generalization relationship).

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

Применительно к диаграммам вариантов использования ассоциация служит для обозначения специфической роли актера при его взаимодействии с отдельным вариантом использования.

На диаграмме вариантов использования, так же как и на других диаграммах, отношение ассоциации обозначается сплошной линией между актером и вариантом использования (см. рис. 1.3).

Отношением зависимости (dependency) является такое отношение между двумя элементами модели, при котором изменение одного элемента (независимого) приводит к изменению другого элемента (зависимого).

Включение (include) в языке UML — это разновидность отношения зависимости между базовым вариантом использования и его специальным случаем.

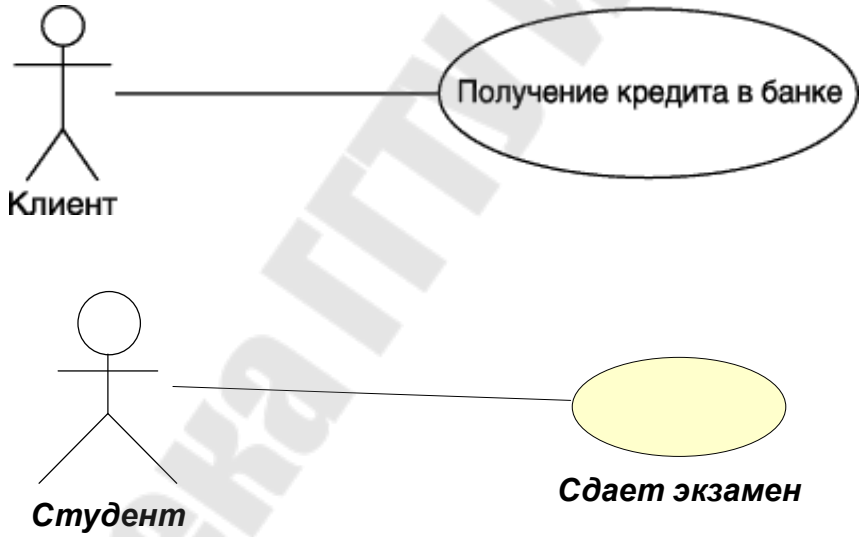

Рис. 1.3. Отношение ассоциации

Отношение включения устанавливается только между двумя вариантами использования и указывает на то, что заданное поведение для одного варианта использования включается в качестве составного фрагмента в последовательность поведения другого варианта использования (рис. 1.4).

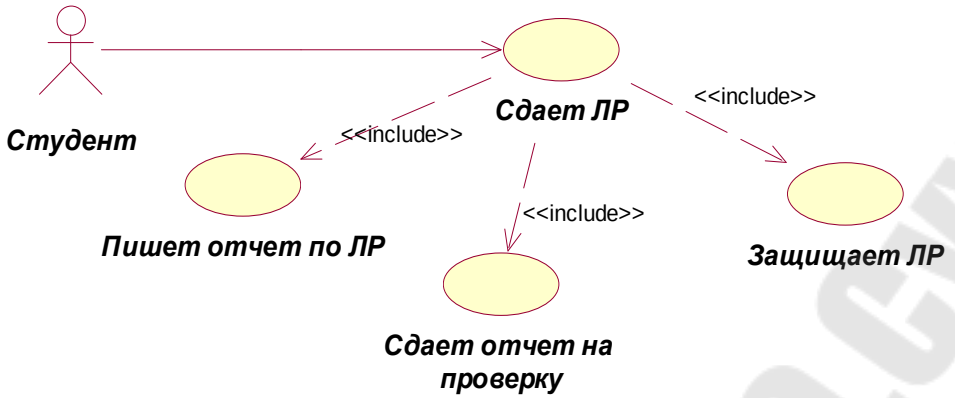

Рис. 1.4. Отношение включения

Процесс выполнения базового варианта использования включает в себя как собственное подмножество последовательность действий, которая определена для включаемого варианта использования. При этом выполнение включаемой последовательности действий происходит всегда при инициировании базового варианта использования.

Отношение расширения (extend) определяет взаимосвязь базового варианта использования с другим вариантом использования, функциональное поведение которого задействуется базовым не всегда, а только при выполнении дополнительных условий.

Отношение расширения между вариантами использования обозначается как отношение зависимости в форме пунктирной линии со стрелкой, направленной от того варианта использования, который является расширением для базового варианта использования.

Данная линия со стрелкой должна быть помечена стереотипом <<extend>> (рис. 1.5).

Рис. 1.5. Отношение расширения

Отношение обобщения служит для указания того факта, что некоторый вариант использования А может быть обобщен до варианта использования В, который называется предком или родителем по отношению А, а вариант А — потомком по отношению к варианту использования В.

Графически данное отношение обозначается сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительский вариант использования.

Эта линия со стрелкой имеет специальное название – стрелка "обобщение".

Отношение обобщения между вариантами использования применяется в том случае, когда необходимо отметить, что дочерние варианты использования обладают всеми атрибутами и особенностями поведения родительских вариантов. Следует учитывать, что дочерние варианты использования участвуют во всех отношениях родительских вариантов.

Пример диаграммы вариантов использования для функциональной модели «Составление расписания» приведен на рис. 1.6.

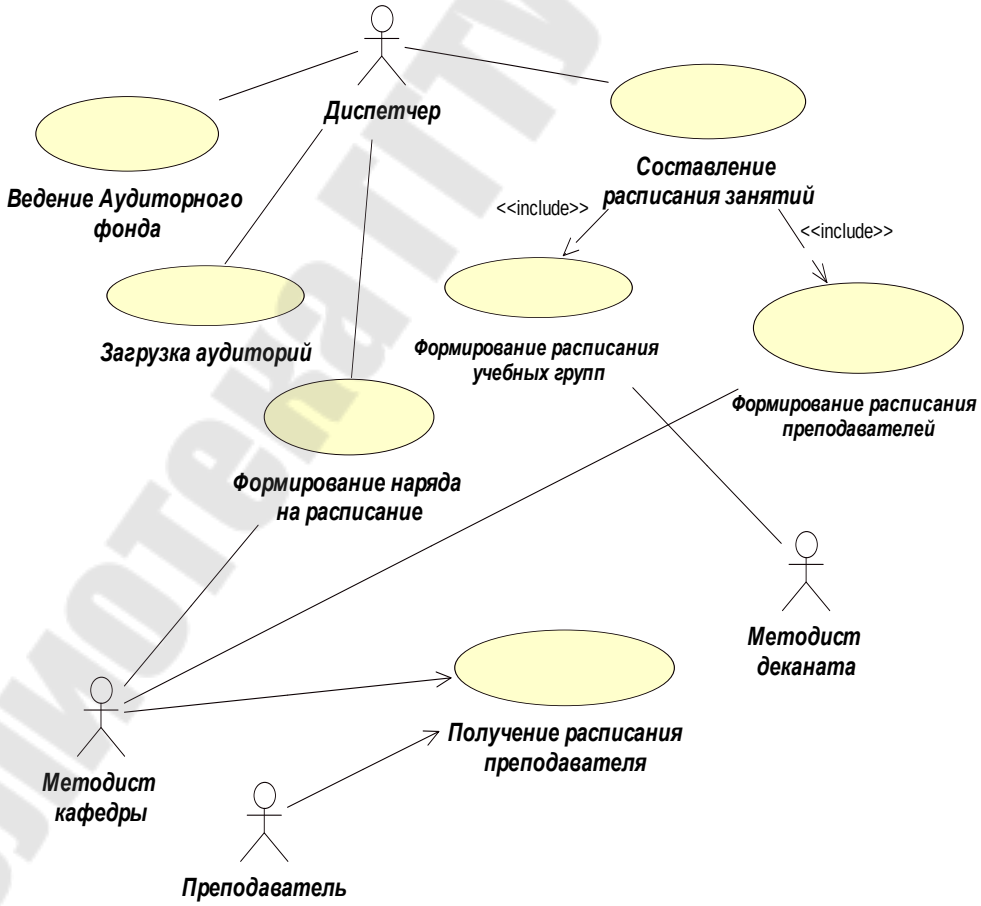

Рис. 1.6. Пример диаграммы вариантов использования

Практическая часть темы 1

Задания для построения диаграммы варианта использования.

1. Выбрать из приложения 1 предметную область для разработки системы автоматизации (СА).

2. Составить для СА краткое техническое задание по следующему плану:

– общие сведения;

– назначение и цели создания СА;

– основные функции СА;

– требования к СА;

– входная информация и выходные формы.

3. Определить актеров и варианты использования и присвоить им имена согласно предметной области.

4. Определить виды связи между ними, выделив ассоциативные связи как основные.

5. Определить связи расширения и включения, если это необходимо при описании функций прецедентов.

6. Построить диаграмму прецедентов по разработанному техническому заданию.

7. Присвоить имя диаграмме согласно предметной области и решаемой задаче.

Тема 2. Диаграмма классов

Краткие теоретические сведения

Диаграмма классов (class diagram) служит для представления статической структуры модели программной системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений.

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов.

Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 2.1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

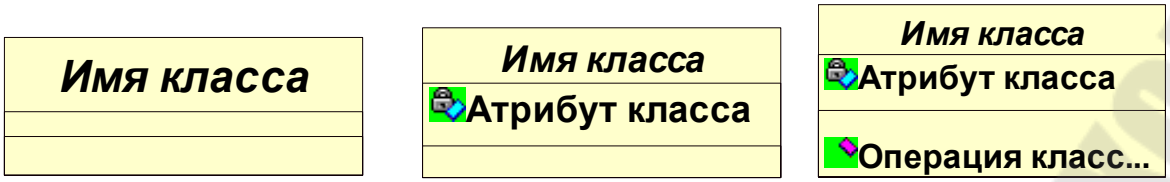

Рис. 2.1. Графическое обозначение класса

Обязательным элементов обозначения класса является его имя. Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Рекомендуется в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов. Необходимо помнить, что именно имена классов образуют словарь предметной области Примерами имен классов могут быть такие существительные, как "Сотрудник", "Компания", "Руководитель", "Клиент", "Преподаватель", "Студент", "Кафедра".

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из:

– квантора видимости атрибута,
– имени атрибута,
– кратности,
– типа значений атрибута,
– исходного значения атрибута.

Квантор видимости может принимать одно из трех возможных значений, приведенных ниже.

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

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

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

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

Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса.

Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса. В качестве примера рассмотрим следующие варианты задания кратности атрибутов:

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

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

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

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

В третьей сверху секции прямоугольника записываются операции или методы класса. Операция (operation) представляет собой некоторый сервис, предоставляющий каждый экземпляр класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса.

Каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения и, возможно, строка-свойство данной операции.

Квантор видимости, как и в случае атрибутов класса, может принимать одно из трех возможных значений.

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

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

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

Кроме внутреннего устройства или структуры классов на соответствующей диаграмме указываются различные отношения между классами.

Отношение ассоциации соответствует наличию некоторого отношения между классами. Данное отношение обозначается сплошной линией с дополнительными специальными символами, которые характеризуют отдельные свойства конкретной ассоциации.

В качестве дополнительных специальных символов могут использоваться имя ассоциации, а также имена и кратность классов-ролей ассоциации. Имя ассоциации является необязательным элементом ее обозначения. Если оно задано, то записывается с заглавной (большой) буквы рядом с линией соответствующей ассоциации.

Отношение обобщения является отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Данное отношение может использоваться для представления взаимосвязей между пакетами, классами, вариантами использования и другими элементами языка UML (рис. 2.2).

Рис. 2.2. Отношение обобщения

На рис. 2.3 приведена диаграмма классов, в которой используются отношения ассоциации («Студент» – «Отчет по ЛР», «Преподаватель» – «Оценки аттестации») и отношения обобщения («Кафедра» – «Преподаватель»).

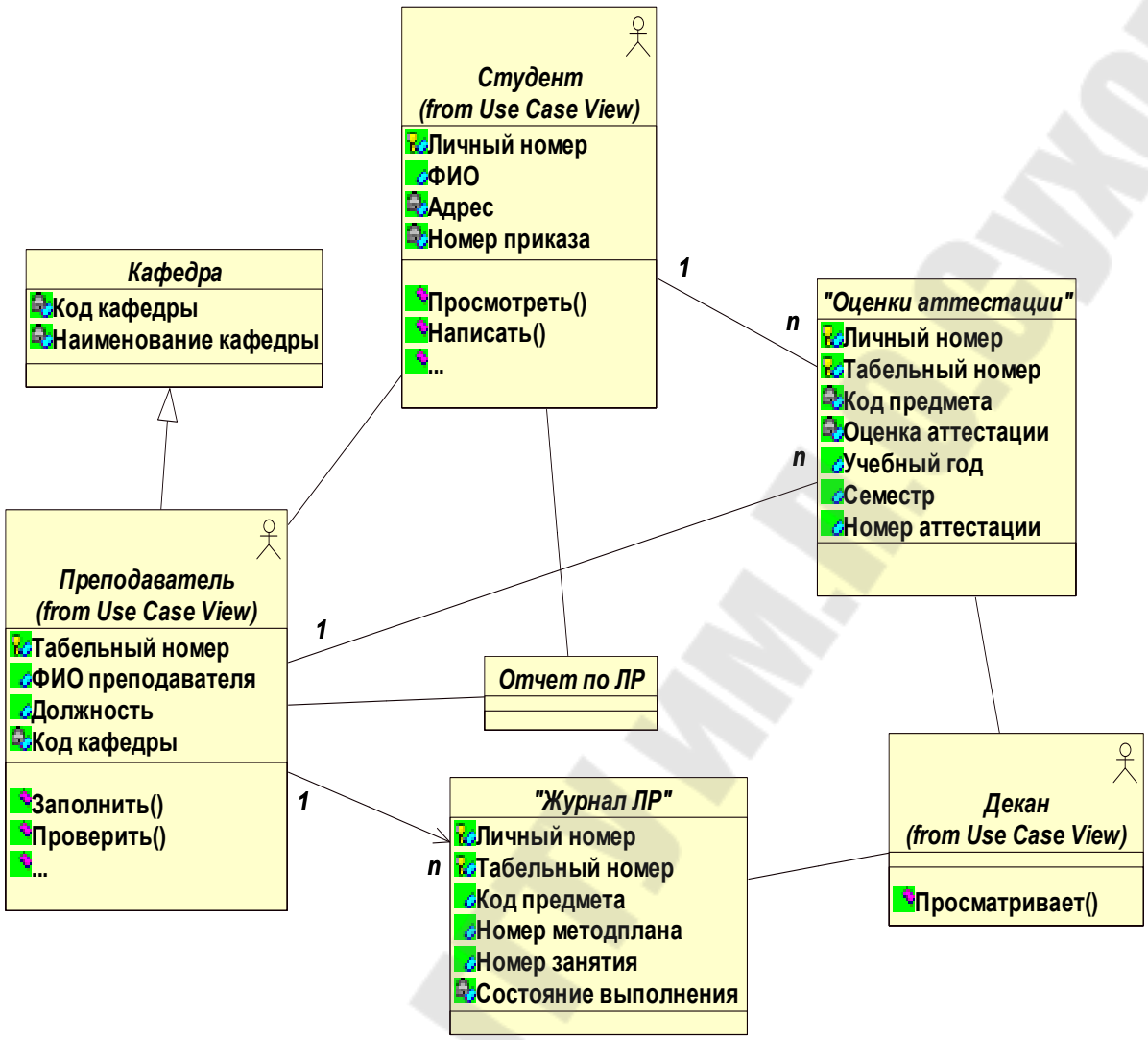

Рис. 2.3. Диаграмма классов с отношениями ассоциации и обобщения

Практическая часть темы 2

Построение диаграммы классов осуществляется на основании диаграммы вариантов использования и технического задания, разработанного для выбранной предметной области (приложение 1).

Задания для построения диаграммы классов.

1. Определить объекты (сущности), привязав их к диаграмме вариантов использования.

2. Дать имя классу для однотипной группы объектов, например объекты «Преподаватели» можно поместить в класс «Преподаватель».

Разработать перечень атрибутов для каждого класса, для каждого атрибута указать:

– квантора видимости атрибута;

– имени атрибута;

– типа значений атрибута;

– исходного значения атрибута (если есть необходимость).

3. Назначить атрибут – ключ (идентификатор объекта), например, для объекта «Преподаватель» – это может быть «Табельный номер преподавателя».

4. Указать операции над классом, например для класса «Преподаватель» – «Проверить()».

5. Построить отношения между классами на основе ассоциаций.

6. Определить направление и множественность, указав нижние и верхние границы.

Тема 3. Диаграммы последовательности и кооперации

Краткие теоретические сведения

Для моделирования взаимодействия объектов в языке UML используются диаграммы взаимодействия. Существует два вида диаграмм взаимодействия:

– диаграммы последовательности (sequence diagrams);

– диаграммы кооперации или сотрудничества (collaboration diagrams).

Диаграммы последовательности определяют временную последовательность передаваемых сообщений, порядок, вид и тип сообщения, происходящих в рамках варианта использования.

Диаграммы последовательности и кооперации являются разными взглядами на одни и те же процессы, поэтому, напримар, Rational Rose позволяет создать из диаграммы последовательности диаграмму кооперации и наоборот, а также производит автоматическую синхронизацию этих диаграмм.

Рассмотрим основные компоненты диаграммы последовательности, одним из которых являются ***объекты***.

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами.

Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. Время

изменяется вдоль вертикального направления, начало отсчета находится вверху.

Диаграмма последовательности имеет два измерения: слева направо размещаются объекты, а сверху вниз отображается время.

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым.

Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый очередностью или степенью активности объектов при взаимодействии друг с другом.

Другим важным компонентом диаграммы последовательности являются ***сообщения***.

Взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения не только передают некоторую информацию, но и требуют или предполагают от принимающего объекта выполнения ожидаемых действий.

Сообщения могут инициировать выполнение операций объектом соответствующего класса, а параметры этих операций передаются вместе с сообщением.

На диаграмме последовательности все сообщения упорядочены по времени своего возникновения в моделируемой системе.

Сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже.

Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Она служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях.

Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней.

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения.

Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы **«X»**.

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосред-

ственно выполняя определенные действия, или состоянии пассивного ожидания сообщений от других объектов.

Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название ***фокуса управления*** (focus of control).

Фокус управления изображается в форме вытянутого узкого прямоугольника, верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а его нижняя сторона – окончание фокуса управления (окончание активности).

Прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни, если на всем ее протяжении он является активным.

На рис. 3.1 приведен графический вид основных компонентов диаграммы последовательности.

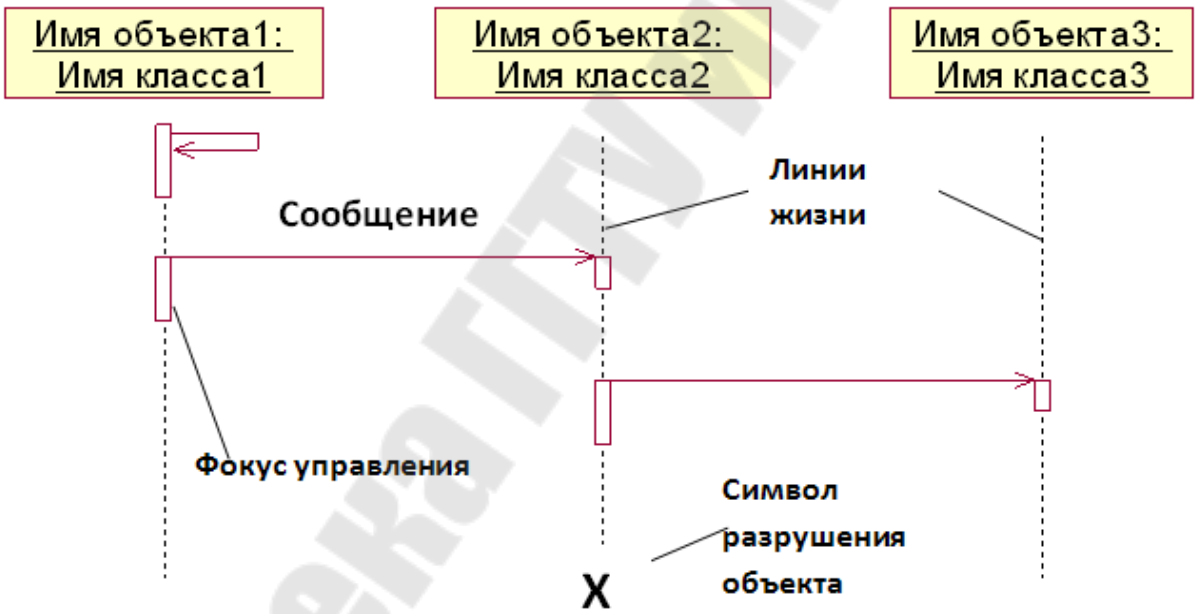

Рис. 3.1. Основные компоненты диаграммы последовательности

На рис. 3.2 приведен пример диаграммы последовательности для задачи аттестации студентов в учебном семестре.

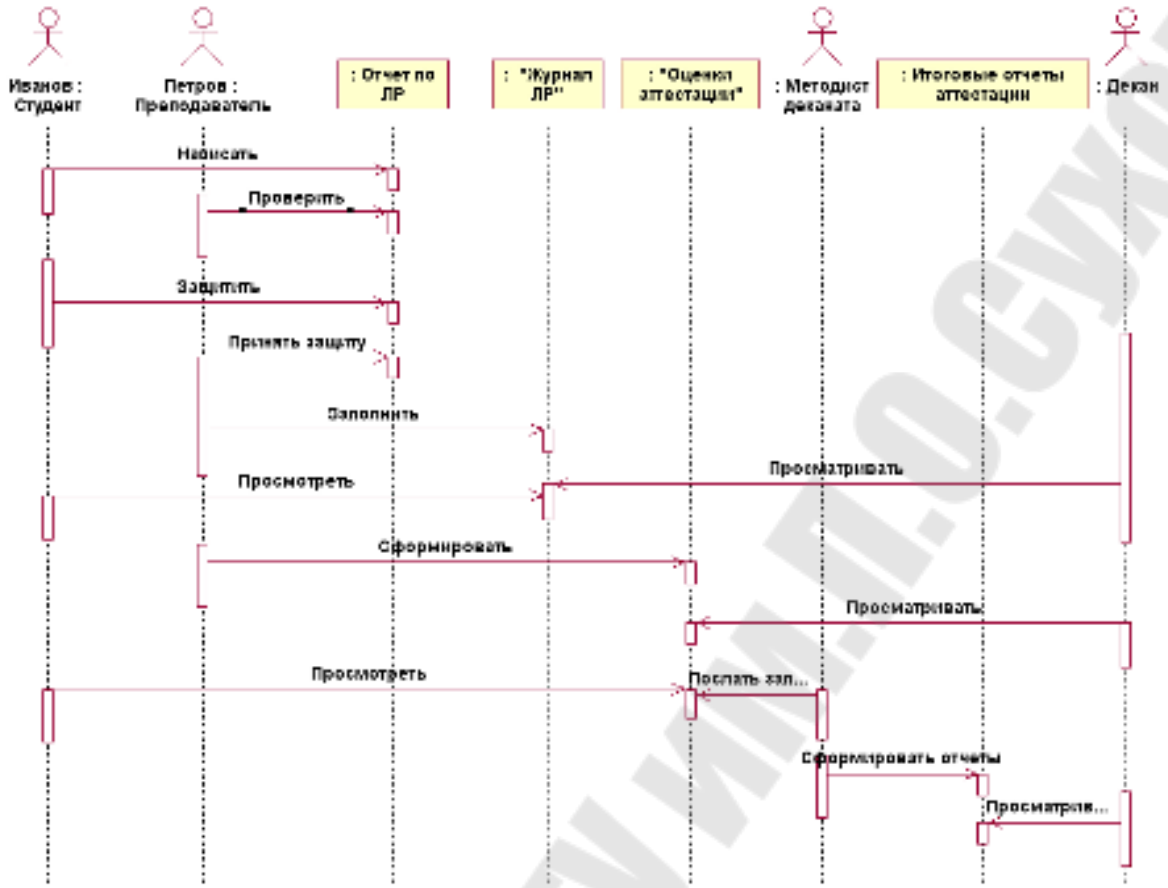

Рис. 3.2. Пример диаграммы последовательности

Диаграмма кооперации предназначена для спецификации структурных аспектов взаимодействия. Главная особенность диаграммы кооперации заключается в возможности графически представить не только последовательность взаимодействия, но и все структурные отношения между объектами, участвующими в этом взаимодействии.

На диаграмме кооперации в виде прямоугольников изображаются участвующие во взаимодействии объекты, содержащие имя объекта, его класс и, возможно, значения атрибутов.

Далее, как и на диаграмме классов, указываются ассоциации между объектами в виде различных соединительных линий. При этом можно явно указать имена ассоциации и ролей, которые играют объекты в данной ассоциации.

Дополнительно могут быть изображены динамические связи — потоки сообщений. Они представляются также в виде соединительных линий между объектами, над которыми располагается стрелка с указанием направления, имени сообщения и порядкового номера в общей последовательности инициализации сообщений

В отличие от диаграммы последовательности, на диаграмме кооперации изображаются только отношения между объектами, играющими определенные роли во взаимодействии.

С другой стороны, на этой диаграмме не указывается время в виде отдельного измерения. Поэтому последовательность взаимодействий и параллельных потоков может быть определена с помощью порядковых номеров.

На рис. 3.3 приведен пример диаграммы кооперации, разработанной на основе диаграммы последовательности, приведенной в п. 4.1.

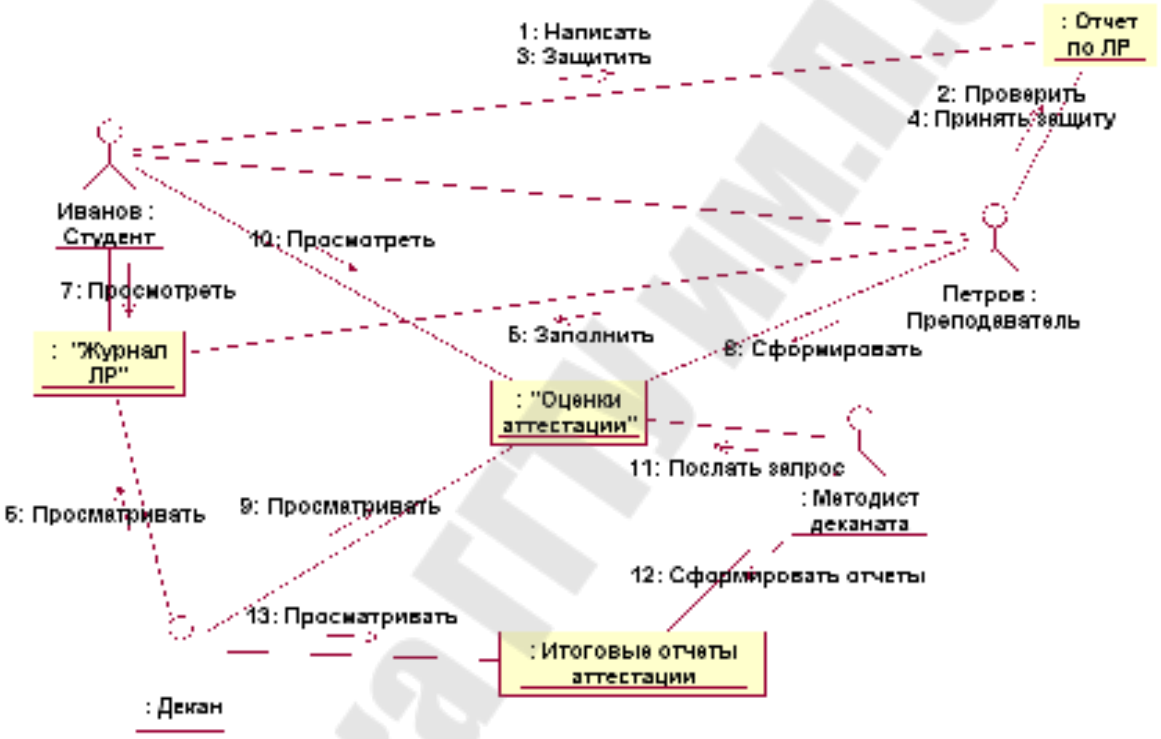

Рис. 3.3. Пример диаграммы кооперации

Практическая часть темы 3

Основой для построения диаграммы последовательности являются разработанные ранее диаграмма вариантов использования и диаграмма классов. Объекты диаграммы последовательности могут относиться к классам, которые уже описаны в диаграмме классов, сообщения могут быть уже описаны как операции классов.

Диаграмма кооперации строится полностью на основе уже разработанной диаграммы последовательности.

Порядок построения диаграмм последовательности и кооперации приведен ниже.

1. Дать имя диаграмме последовательности.

2. Определить объекты, привязав их к диаграмме классов и вариантов использования.

3. Создать линии жизни объектов.

4. Установить сообщения между объектами, используя там, где возможно, операции классов.

5. Присвоить имена новым сообщениям.

6. Пронумеровать сообщения.

7. Выбрать все объекты диаграммы последовательности для построения диаграммы кооперации.

8. Соединить объекты направленными стрелками с указанием номера и имени сообщения.

Тема 4. Диаграммы состояний

Краткие теоретические сведения

Для моделирования поведения на логическом уровне в языке UML могут использоваться канонические диаграммы **состояний и деятельности**.

Каждая из них фиксирует внимание на отдельном аспекте функционирования системы.

В отличие от других диаграмм диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее — одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта.

Главное предназначение этой диаграммы — описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла.

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. Основными понятиями, входящими в формализм автомата, являются состояние и переход.

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия.

Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта (рис. 4.1).

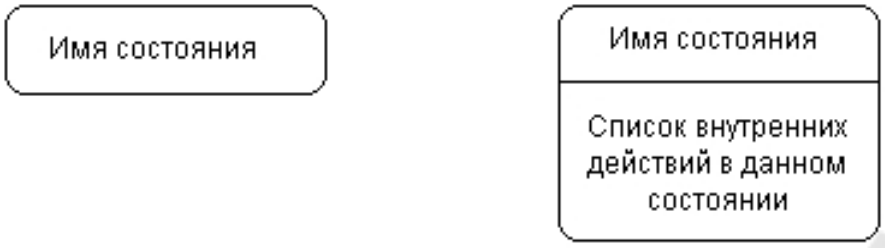

Рис. 4.1. Графическое обозначение состояния

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий. В этом состоянии находится объект по умолчанию в начальный момент времени.

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий. В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени (рис. 4.2).

Рис. 4.2. Графическое обозначение начального и конечного состояний

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим.

Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий.

В этом случае говорят, что переход срабатывает, или происходит срабатывание перехода.

До срабатывания перехода объект находится в предыдущем от него состоянии, называемым исходным состоянием, или в источнике, а после его срабатывания объект находится в последующем от него состоянии (целевом состоянии).

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала.

На переходе указывается: имя события; действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое.

Срабатывание перехода может зависеть не только от наступления некоторого события, но и от выполнения определенного условия, называемого **сторожевым условием**.

Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение "истина".

Событие представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени.

Про события говорят, что они "происходят", при этом отдельные события должны быть упорядочены во времени.

После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

В языке UML события играют роль стимулов, которые инициируют переходы из одних состояний в другие.

В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий.

Имя события идентифицирует каждый отдельный переход на диаграмме состояний и может содержать строку текста, начинающуюся со строчной буквы.

В этом случае принято считать переход **триггерным**, т. е. таким, который специфицирует событие-триггер.

Если рядом со стрелкой перехода не указана никакая строка текста, то соответствующий переход является **нетриггерным**, и в этом случае из контекста диаграммы состояний должно быть ясно, после окончания какой деятельности он срабатывает.

После имени события могут следовать круглые скобки для явного задания параметров соответствующего события-триггера. Если таких параметров нет, то список параметров со скобками может отсутствовать.

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

Если сторожевое условие принимает значение "истина", то соответствующий переход может сработать, в результате чего объект перейдет в целевое состояние.

Пример диаграммы состояний приведен на рис. 4.3.

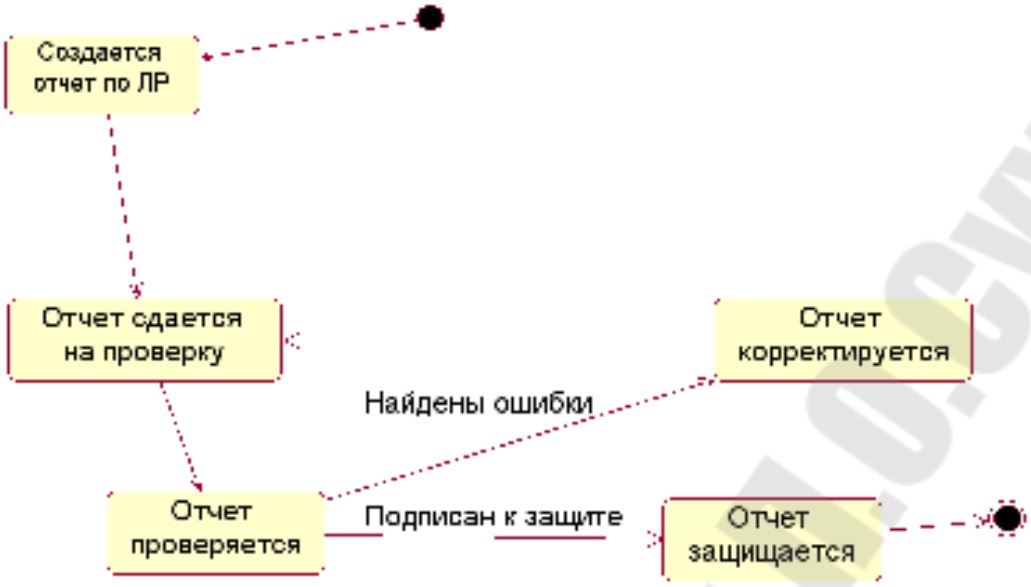

Рис. 4.3. Пример диаграммы состояний

Практическая часть темы 4

Построение диаграммы состояния осуществляется на основе разработанных ранее диаграмм классов и диаграммы вариантов использования. Как правило, для диаграммы состояний выбирается один объект или объекты одного класса, для которых необходимо проследить последовательность и условия изменения состояний от начала формирования объекта до окончания его функционирования в данном процессе.

Последовательность разработки диаграммы состояний приведена ниже.

1. Дать имя диаграмме состояний.
2. Выбрать классы, для объектов которых будет строиться диаграмма состояний.
3. Разработать перечень состояний для объекта, учитывая начальное и конечное состояния.
4. Соединить состояния направленными переходами.
5. Выделить триггерные и нетриггерные переходы.
6. Разработать для триггерных переходов сторожевые условия и события.
7. Для каждого выбранного класса объектов разработать собственную диаграмму состояний.

Тема 5. Диаграммы деятельности

Краткие теоретические сведения

При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций.

Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата.

Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий.

Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой, операции в предыдущем состоянии.

Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами — переходы от одного состояния действия к другому. Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний.

Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения.

При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом.

Отдельные элементарные вычисления могут приводить к некоторому результату или действию (action).

На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности.

Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и по крайней мере одним выходящим из состояния переходом.

Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным.

Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления (рис. 5.1).

Рис. 5.1. Примеры состояний действия:
простое действие и выражение

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

Если из состояния действия выходит единственный переход, то он может быть никак не помечен.

Если же таких переходов несколько, то сработать может только один из них. Именно в этом случае для каждого из таких переходов должно быть явно записано сторожевое условие в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них.

Если последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата, то в диаграмме применяется символ ветвления.

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений.

Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем, необходимы графические примитивы для представления параллельных процессов.

В языке UML для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления. Таким символом является прямая черточка, аналогично обозначению перехода в формализме сетей Петри (рис. 5.2).

На рис. 5.3 приведен пример диаграммы деятельности с ветвлением и слиянием.

Рис. 5.2. Примеры разделения и слияния в диаграммах деятельности

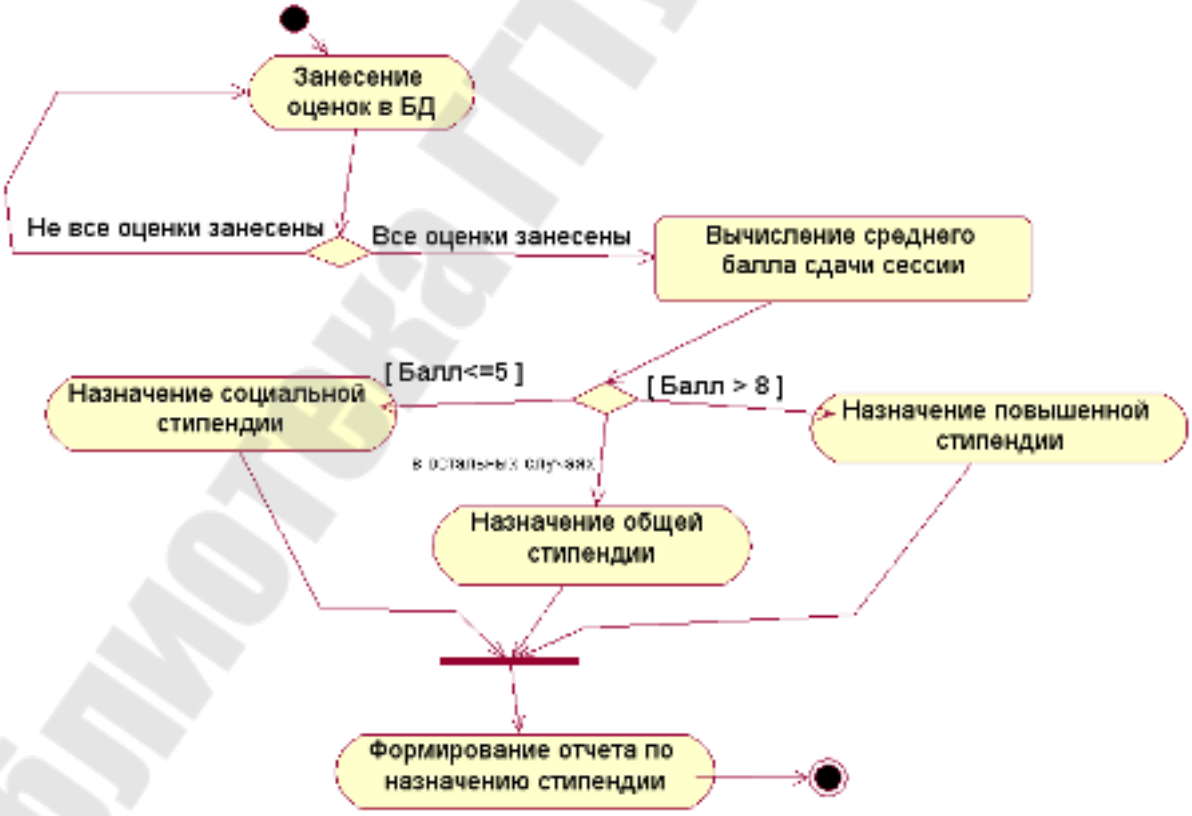

Рис. 5.3. Диаграмма деятельности с ветвлением и слиянием

Применительно к бизнес-процессам желательно выполнение каждого действия ассоциировать с конкретным подразделением компании. В этом случае подразделение несет ответственность за реализацию отдельных действий, а сам бизнес-процесс представляется в виде переходов действий из одного подразделения к другому.

Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название **дорожки (swimlanes).**

Все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании. Названия подразделений явно указываются в верхней части дорожки.

Пересекать линию дорожки могут только переходы, которые в этом случае обозначают выход или вход потока управления в соответствующее подразделение компании.

Порядок следования дорожек не несет какой-либо семантической информации и определяется соображениями удобства.

В качестве примера рассмотрим фрагмент диаграммы деятельности при подготовке и защите курсовых работ. Подразделениями, участвующими в процессе, являются студент, преподаватель, комиссия по защите и кафедра (секретарь).

Этим подразделениям будут соответствовать четыре дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения.

В данном случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое из подразделений должно выполнять то или иное действие (рис. 5.4).

Практическая часть темы 5

Диаграмма активности строится на основе диаграммы классов и диаграммы состояния, разработанных ранее. Кроме того, в диаграмме могут участвовать наименования подразделений предприятия и организации, для которой строится автоматизированная система.

1. Дать имя диаграмме деятельности.

2. Выбрать подразделения, для объектов которых будет строиться диаграмма деятельности.

3. Выделить зоны ответственности каждого подразделения и указать наименования подразделений на дорожках.

4. Для каждой дорожки разработать набор состояний действия.

5. Выделить ветвления и разработать переходы с использованием ветвлений.

6. Указать сторожевые условия на переходах, там, где это необходимо.

7. Выделить параллельные процессы и соединить нужные состояния действия с использованием графических символов раздела и слияния.

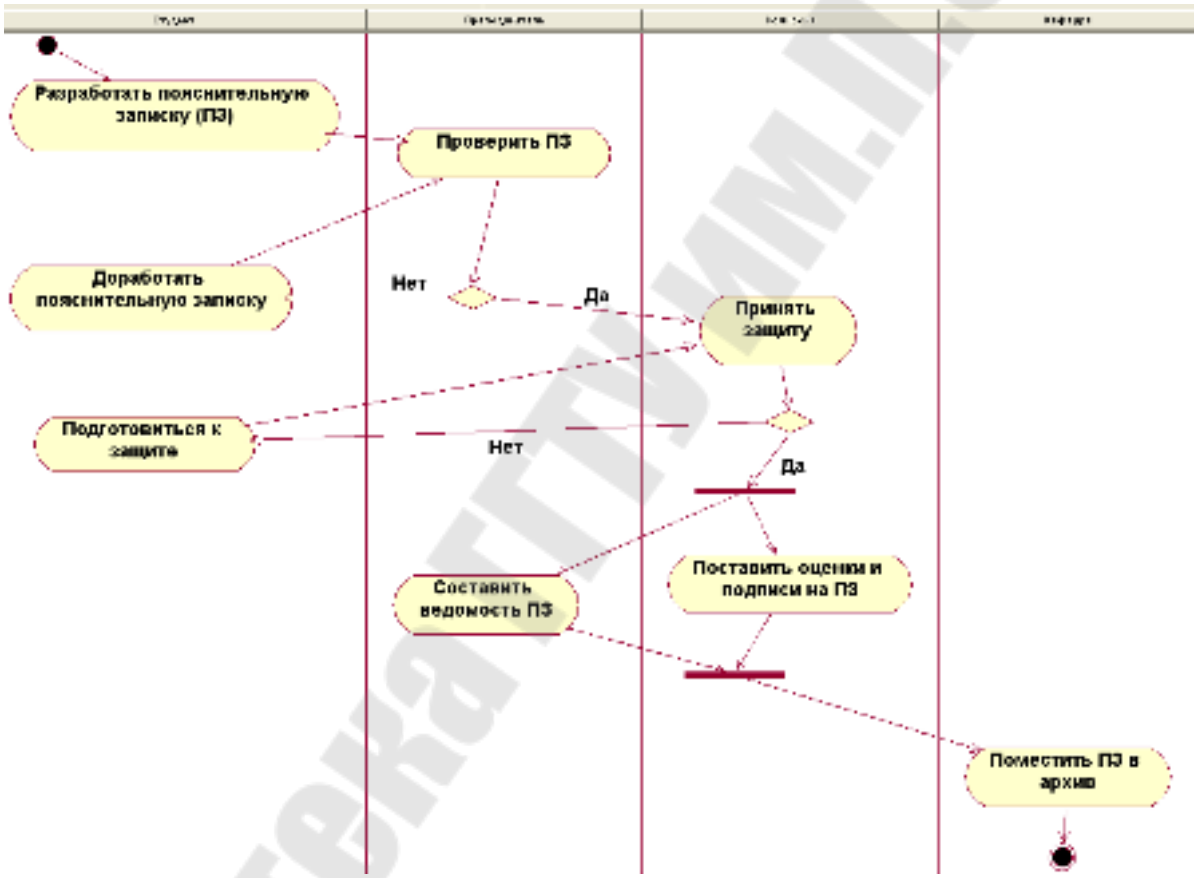

Рис. 5.4. Диаграмма деятельности с применением дорожек

Литература

1. Ларман К. Применение UML 2.0 и шаблонов проектирования : пер. с англ. – 3-е изд. – М.: Вильямс, 2007.

2. Фаулер М., Скотт К. UML. Основы. – СПб.: Символ-Плюс, 2002.

3. Кватрани Т. Визуальное моделирование с помощью Rational Rose 2002 и UML.– М.: Вильямс, 2003.

4. Рамбо Дж. UML: Специальный справочник. – СПб.: Питер, 2002.

Приложение

Список подсистем автоматизации для разработки диаграмм UML

Вариант 1

Подсистема автоматизированной записи студентов различных факультетов и курсов на изучение различных дисциплин по выбору студентов с последующим формированием групп.

Вариант 2

Подсистема автоматизированного распределения лекционных курсов по различным дисциплинам между преподавателями различных кафедр различных факультетов.

Вариант 3

Подсистема автоматизированной регистрации участников конференции из различных ВУЗов различных городов и стран на различные секции конференции с выступлением или без него.

Вариант 4

Автоматизированная подсистема распределения выпускников различных специальностей университета на различные предприятия различных областей и городов с различными должностями и окладами в зависимости от среднего балла выпускника.

Вариант 5

Автоматизированная подсистема контроля сдачи экзаменов и зачисления абитуриентов в университет с формированием учебных групп.

Вариант 6

Автоматизированная подсистема ведения личных дел студентов в студенческом отделе кадров, с регистрацией поступления, перевода на новый курс, защиты диплома и распределения на работу.

Вариант 7

Автоматизированная подсистема учета оплаты за обучение по договорам студентов различных групп и факультетов дневной и заочной форм обучения.

Вариант 8

Автоматизированная подсистема дистанционного обучения по какой-либо дисциплине студента какой-либо группы какого-либо факультета университета.

Вариант 9

Автоматизированная подсистема формирования и выдачи пакетов учебных пособий студентам различных групп различных факультетов университета.

Вариант 10

Автоматизированная подсистема распределения мест в студенческом общежитии и ведения списка студентов, проживающих в общежитии.

Вариант 11

Автоматизированная подсистема составления расписания учебных занятий для групп слушателей ИПК.

Вариант 12

Автоматизированная подсистема учета успеваемости слушателей ИПК.

Вариант 13

Автоматизированная подсистема ведения дипломного проектирования слушателей ИПК.

Вариант 14

Автоматизированная подсистема ведения курсового проектирования слушателей ИПК.

Содержание

Трохова Татьяна Анатольевна

ФУНКЦИОНАЛЬНОЕ МОДЕЛИРОВАНИЕ ПРОГРАММНЫХ СИСТЕМ В UML

Методические указания по курсу «Технологии проектирования программного обеспечения информационных систем» для слушателей специальности 1-40 01 73 «Программное обеспечение информационных систем»

Подписано в печать 24.10.12.
Формат 60x84/16. Бумага офсетная. Гарнитура «Таймс».
Ризография. Усл. печ. л. 2,09. Уч.-изд. л. 1,99.
Изд. № 1.
http://www.gstu.by

Отпечатано на цифровом дуплекаторе
с макета оригинала авторского.
Учреждение образования «Гомельский государственный
технический университет имени П.О. Сухого».
246746, г. Гомель, пр. Октября, 48.