ЛАБОРАТОРНАЯ РАБОТА № 1

ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Цель работы: закрепление теоретических сведений о диаграмме вариантов использования; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме вариантов использования.

Краткие теоретические сведения

Диаграммы вариантов использования описывают функциональное назначение системы или то, что система должна делать в процессе своего функционирования.

Диаграмма вариантов использования (use case diagram) — диаграмма, на которой изображаются отношения между актерами и вариантами использования.

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 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.

Назначение диаграммы: проектируемая программная система представляется в форме вариантов использования, с которыми взаимодействуют внешние сущности или актеры.

Диаграмма вариантов использования представляет собой граф специального вида, который является графической нотацией для представления конкретных вариантов использования, актеров и отношений между этими элементами.

Базовыми элементами диаграммы вариантов использования являются вариант использования и актер.

Вариант использования (use case) — внешняя спецификация последовательности действий, которые система или другая сущность могут выполнять в процессе взаимодействия с актерами.

Вариант использования представляет собой спецификацию общих особенностей поведения или функционирования моделируемой системы без рассмотрения внутренней структуры этой системы.

Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером, сами эти действия не изображаются на рассматриваемой диаграмме.

Содержание варианта использования может быть представлено в форме дополнительного пояснительного текста, который раскрывает смысл или семантику действий при выполнении данного варианта использования. Такой пояснительный текст получил название *текста-сценария* или просто *сценария*.

Вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его имя в форме существительного (рис. 2, а) или глагола (рис. 2, б) с пояснительными словами. Сам текст имени должен начинаться с заглавной буквы.

Имя - строка текста, которая используется для идентификации любого элемента модели.

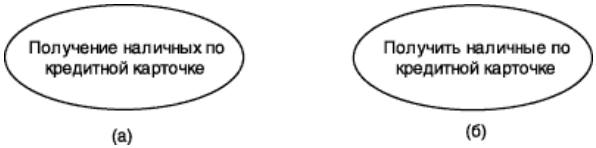


Рис. 2 Графическое обозначение варианта использования

Каждый вариант использования соответствует отдельному сервису, который предоставляет моделируемая система по запросу актера, т. е. определяет один из способов применения системы. Сервис, который инициализируется по запросу актера, должен представлять собой законченную последовательность действий (после того как система закончит обработку запроса актера, она должна возвратиться в исходное состояние, в котором снова готова к выполнению следующих запросов).

Диаграмма вариантов использования содержит конечное множество вариантов использования, которые в целом должны определять все возможные стороны ожидаемого поведения системы. Для удобства множество вариантов использования может рассматриваться как отдельный пакет.

Актер (actor) — согласованное множество ролей, которые играют внешние сущности по отношению к вариантам использования при взаимодействии с ними.

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 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 (рис. 3).

Рис. 3 Графическое обозначение актера

В некоторых случаях актер может обозначаться в виде прямоугольника класса со стереотипом <<actor>> и обычными составляющими элементами класса. Имена актеров должны начинаться с заглавной буквы и следовать рекомендациям использования имен для типов и классов модели. При этом символ отдельного актера связывает соответствующее описание актера с конкретным именем.

Актеры используются для моделирования внешних по отношению к проектируемой системе сущностей, которые взаимодействуют с системой. В качестве актеров могут выступать другие системы, в том числе подсистемы проектируемой сис-

темы или ее отдельные классы. Наиболее наглядный пример актера — конкретный посетитель web-сайта в Интернет со своими параметрами аутентификации.

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 (например, датчик, устройство считывания штрих-кода и т.д.).

Интерфейс обозначается в виде маленького круга, рядом с которым записывается его имя. В качестве имени может быть существительное или строка текста. Если имя записывается на английском языке, то оно начинается с заглавной буквы I.

Графический символ интерфейса соединяется с тем вариантом использования, который его поддерживает, на диаграмме двумя способами:

- сплошной линией (указывает, что связанный с интерфейсом вариант использования должен реализовывать все необходимые для него сервисы);

- пунктирной линией со стрелкой (означает, что вариант использования предназначен для спецификации только того сервиса, который необходим для реализации данного интерфейса).

Примечание (note) предназначено для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта (комментарии разработчика - дата; ограничения - на значения отдельных связей или экземпляры сущностей; помеченные значения).

Графически обозначаются прямоугольником с "загнутым" верхним правым уголком (рис. 4). Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия. Если примечание относится к нескольким элементам, то от него проводятся несколько линий.

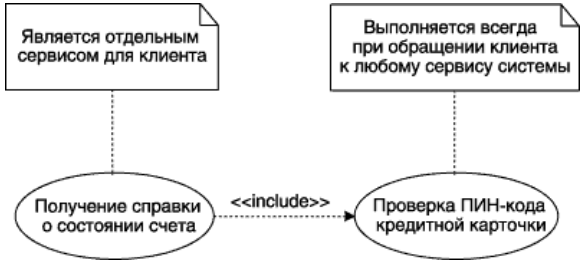


Рис. 4. Примеры примечаний на диаграммах вариантов использования

Отношение (relationship) — семантическая связь между отдельными элементами модели.

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 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).

Общие свойства вариантов использования могут быть представлены тремя различными способами: с помощью отношений включения, расширения и обобщения.

Отношение ассоциации используется при построении всех графических моделей систем. Применительно к диаграммам вариантов использования ассоциация служит для обозначения конкретной 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. Эта линия может иметь некоторые дополнительные обозначения, например, имя и кратность (рис. 5).

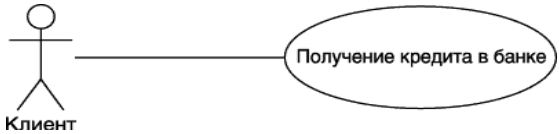


Рис. 5 Пример графического представления отношения ассоциации между актером и вариантом использования

Отношение включения (include) — это разновидность отношения зависимости между базовым вариантом использования и его специальным случаем.

Отношение включения устанавливается 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.

Графически отношение включения обозначается в форме пунктирной линии со стрелкой, направленной от базового варианта использования к включаемому варианту использования. При этом данная линия помечается стереотипом <<include>>, как показано на рис. 6.

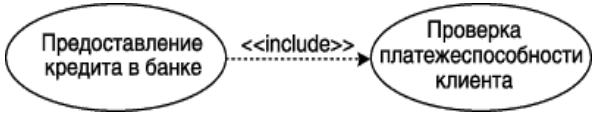


Рис. 6. Пример графического изображения отношения включения между вариантами использования

Отношение включения, направленное от варианта использования "Предоставление кредита в банке" к варианту использования "Проверка платежеспособности клиента", указывает на то, что каждый экземпляр первого варианта использования всегда включает в себя функциональное поведение или выполнение второго варианта использования. В этом смысле поведение второго варианта использования является частью поведения первого варианта использования на данной диаграмме.

Один вариант использования может входить в несколько других вариантов, а также содержать в себе другие варианты. Включаемый вариант использования является независимым от базового варианта в том смысле, что он предоставляет последнему инкапсулированное поведение, детали реализации которого скрыты от последнего и могут быть легко перераспределены между несколькими включаемыми вариантами использования. Базовый вариант зависит только от результатов выполнения включаемого в него варианта использования, но не от структуры включаемых в него вариантов.

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.

Графически обозначается 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>>, как показано на рис. 7.

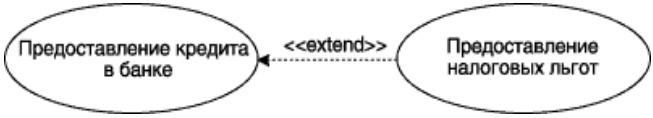


Рис. 7. Пример графического изображения отношения расширения между вариантами использования

В изображенном фрагменте имеет место отношение расширения между базовым вариантом использования "Предоставление кредита в банке" и вариантом использования "Предоставление налоговых льгот". Это означает, что свойства поведения первого варианта использования в некоторых случаях могут быть дополнены функциональностью второго варианта использования. Для того чтобы это расширение имело место, должно быть выполнено определенное логическое условие данного отношения расширения.

Если базовый вариант использования выполняет некоторую последовательность действий, которая определяет его поведение, и при этом имеется точка расширения на экземпляр другого варианта использования, то проверяется логическое условие данного отношения. Если это условие выполняется, исходная последовательность действий расширяется посредством включения действий другого варианта использования. Условие отношения расширения проверяется один раз — при первой ссылке на точку расширения, и если оно выполняется, то все расширяющие варианты использования вставляются в базовый вариант.

Отношение обобщения используется тогда, когда два и более актера могут иметь общие свойства, т. е. взаимодействовать с одним и тем же множеством вариантов использования одинаковым образом. Такая общность свойств и поведения представляется в виде отношения обобщения с другим актером, который моделирует соответствующую общность ролей.

Графически обозначается сплошной линией со стрелкой в форме не закрашенного треугольника, которая указывает на родительский вариант использования (рис. 8). Эта линия со стрелкой имеет специальное название — стрелка-обобщение.

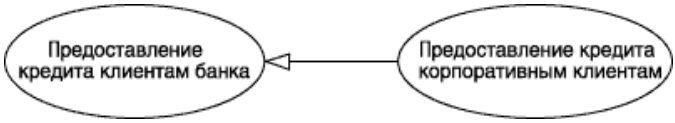


Рис. 8. Пример графического изображения отношения обобщения между вариантами использования

В данном примере отношение обобщения указывает на то, что вариант использования "Предоставление кредита корпоративным клиентам" - специальный случай варианта использования "Предоставление кредита клиентам банка". Другими словами, первый вариант использования является специализацией второго варианта использования.

Вариант использования "Предоставление кредита клиентам банка" еще называют *предком* или родителем по отношению к варианту использования "Предоставление кредита корпоративным клиентам", а последний - *потомком* по отношению к первому варианту использования.

Потомок наследует все свойства поведения своего родителя, а также может обладать дополнительными особенностями поведения.

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

Язык UML включает в себя еще и дополнительные графические обозначения, ориентированные для решения задач из определенной предметной области. Примеры подобных обозначений, которые используются для моделирования бизнес-систем и могут быть изображены на диаграммах вариантов использования: бизнес-актер, сотрудник и бизнес - вариант использования.

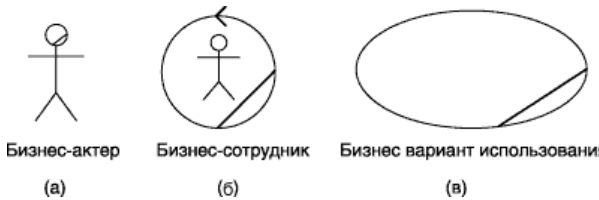


Рис. 9. Графические изображения бизнес-актера (а), бизнес-сотрудника (б) и бизнес-варианта использования (в)

Применение этих элементов графической нотации иллюстрирует пример представления диаграмм вариантов использования для системы продажи товаров в супермаркете.

Одно из главных назначений диаграммы вариантов использования заключается в формализации функциональных требований к системе. Общее количество актеров в модели не должно превышать 20, а вариантов использования -50, в противном случае модель теряет свою наглядность.

Для разработки диаграммы вариантов использования рекомендуется некоторая последовательность действий:

- определить главных и второстепенных актеров;
- определить цели главных актеров по отношению к системе;
- сформулировать основные варианты использования, которые специфицируют функциональные требования к системе;
- упорядочить варианты использования по степени убывания риска их реализации;
- рассмотреть все базовые варианты использования в порядке убывания их степени риска;
- выделить участников, интересы, условия выполнения выбранного варианта использования;
- написать сценарий реализации выбранного варианта использования;
- определить исключения в выполнении сценария варианта использования;
- написать сценарии для всех исключений;
- выделить общие варианты использования и изобразить их взаимосвязи с базовыми со стереотипом <<include>>;
- выделить варианты использования для исключений и изобразить их взаимосвязи с базовыми со стереотипом <<extend>>;
- проверить диаграмму на отсутствие дублирования вариантов использования и актеров.

Пример: Программное средство представляет собой базу данных «Автоматизация процесса составления расписания в учебном заведении». Программное средство обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствие с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

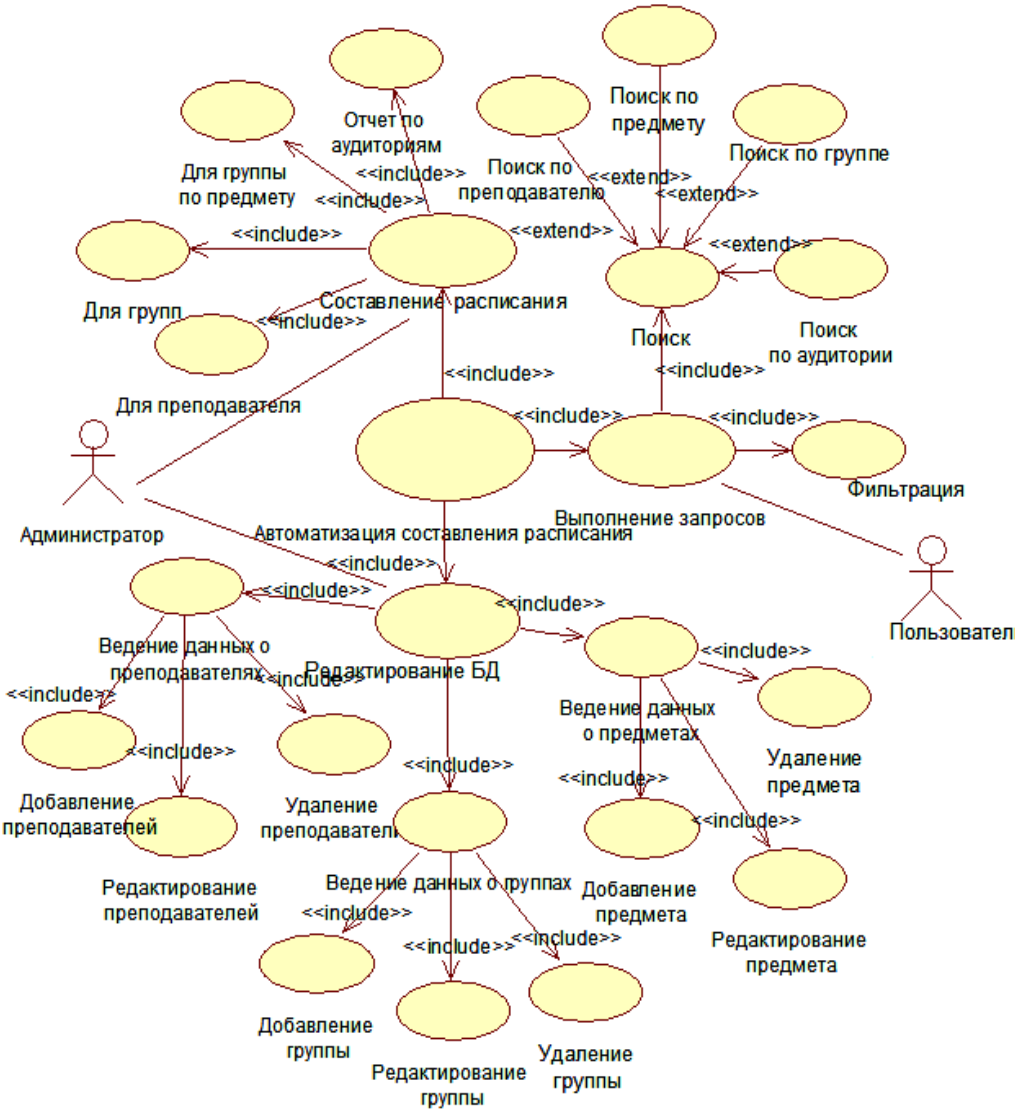


Рис. 10. Пример диаграммы вариантов использования

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Диаграмма вариантов использования”.
2. Разработать диаграмму вариантов использования для произвольной системы своего индивидуального задания (Приложение А).

3. Оформить отчет, включив в него описание всех компонентов диаграммы вариантов использования согласно индивидуальному варианту задания, ответы на контрольные вопросы.

Контрольные вопросы

1. Дайте определение понятию «диаграмма вариантов использования».
2. Опишите назначение диаграммы вариантов использования.
3. Дайте определение следующим понятиям «актер», «вариант использования», «интерфейс», «сценарий».
4. Дайте определение и классификацию отношений в диаграмме вариантов использования.
5. В чем заключается отношение включения?
6. В чем заключается отношение ассоциации?
7. В чем заключается отношение обобщения?
8. В чем заключается отношение расширения?

ЛАБОРАТОРНАЯ РАБОТА № 2
ДИАГРАММА КЛАССОВ

Цель работы: закрепление теоретических сведений о диаграмме классов; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме классов.

Краткие теоретические сведения

Диаграмма классов отражает различные взаимосвязи между отдельными сущностями предметной области, а также описывает их внутреннюю структуру и типы отношений.

Диаграмма классов (class diagram) — диаграмма языка UML, на которой представлена совокупность декларативных или статических элементов модели, таких как классы с атрибутами и операциями, а также связывающие их отношения.

Диаграмма классов предназначена для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования.

Диаграмма классов содержит интерфейсы, пакеты, отношения и даже отдельные экземпляры классификаторов, такие как объекты и связи.

Класс (class) — абстрактное описание множества однородных объектов, имеющих одинаковые атрибуты, операции и отношения с объектами других классов.

Графически изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 11). В этих секциях могут указываться имя класса, атрибуты и операции класса. Даже если секции атрибутов и операций пусты, в обозначении класса они должны быть выделены горизонтальной линией.

Рис. 11. Варианты графического изображения класса на диаграмме классов

В первом случае для класса Окружность (рис. 12, а) указаны только его атрибуты – точка на координатной плоскости, которая определяет расположение ее центра. Для класса Окно (рис. 12, б) указаны только его операции, при этом секция его атрибутов оставлена пустой. Для класса Счет (рис. 12, в) дополнительно изображена четвертая секция, в которой указано требование – реализовать резервное копирование объектов этого класса.

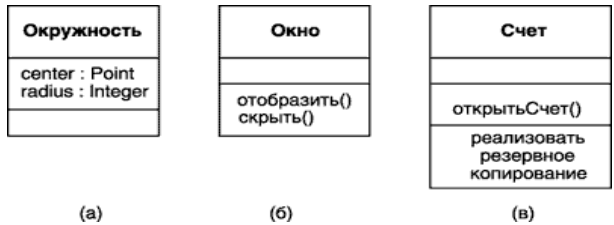


Рис. 12. Примеры графического изображения конкретных классов

Имя класса должно быть уникальным в пределах пакета, который может содержать одну или несколько диаграмм классов. Имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы.

В секции имени класса могут также находиться стереотипы или ссылки на стандартные шаблоны, от которых образован данный класс и, соответственно, от которых он наследует атрибуты и операции. Здесь также могут записываться и другие общие свойства этого класса.

Класс может иметь или не иметь экземпляров или объектов. В зависимости от этого в языке UML различают конкретные и абстрактные классы.

Конкретный класс (concrete class) — класс, на основе которого могут быть непосредственно созданы экземпляры или объекты.

Рассмотренные выше обозначения относятся к конкретным классам.

Абстрактный класс (abstract class) — класс, который не имеет экземпляров или объектов.

Для обозначения имени абстрактного класса используется курсив.

Когда необходимо явно указать, к какому пакету относится тот или иной класс, используется специальный символ разделитель – двойное двоеточие - (::). Синтаксис строки имени класса в этом случае будет следующий:

<Имя пакета>::<Имя класса>

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

Атрибут (attribute) — содержательная характеристика класса, описывающая множество значений, которые могут принимать отдельные объекты этого класса.

Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса. Каждому атрибуту класса соответствует отдельная строка текста, которая состоит из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения.

Общий формат записи отдельного атрибута класса следующий:

<квантор видимости> <имя атрибута> [кратность] : <тип атрибута> = <исходное значение> {строка-свойство}

Видимость (visibility) — качественная характеристика описания элементов класса, характеризующая потенциальную возможность других объектов модели оказывать влияние на отдельные аспекты поведения данного класса.

Видимость в языке UML специфицируется с помощью *квантора видимости* (может быть опущен), который может принимать одно из 4-х возможных значений и отображаться при помощи специальных символов.

- "+" – область видимости типа общедоступный (public). Атрибут с этой областью видимости доступен или виден из любого другого класса пакета, в котором определена диаграмма.
- "#" – область видимости типа защищенный (protected). Атрибут недоступен или невиден для всех классов, за исключением подклассов данного класса.
- "-" – область видимости типа закрытый (private). Атрибут недоступен или невиден для всех классов без исключения.
- "~" – область видимости типа пакетный (package). Атрибут недоступен или невиден для всех классов за пределами пакета, в котором определен класс-владелец данного атрибута.

Вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private, package.

Имя атрибута используется в качестве идентификатора соответствующего атрибута и поэтому должно быть уникальным в пределах данного класса. Имя атрибута - обязательный элемент, должно начинаться со строчной (малой) буквы и не должно содержать пробелов.

Кратность (multiplicity) — спецификация области значений допустимой мощности, которой могут обладать соответствующие множества.

Кратность атрибута характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса:

[нижняя граница .. верхняя граница].

Интервалов кратности для отдельного атрибута может быть несколько. Если в качестве кратности указывается единственное число, то кратность атрибута принимается равной данному числу. Если же указывается "*", то кратность атрибута может быть произвольным положительным целым числом или нулем.

Тип атрибута представляет собой выражение, семантика которого определяется некоторым типом данных, определенным в пакете Типы данных языка UML или самим разработчиком, или в зависимости от языка программирования, который предполагается использовать для реализации данной модели.

Исходное значение служит для задания начального значения соответствующего атрибута в момент создания отдельного экземпляра класса. Если оно не указано, то значение соответствующего атрибута не определено на момент создания нового экземпляра класса.

При задании атрибутов могут быть использованы дополнительные синтаксические конструкции — это подчеркивание строки атрибута, пояснительный текст в фигурных скобках и косая черта перед именем атрибута. Подчеркивание строки атрибута означает, что соответствующий атрибут общий для всех объектов данного класса, т.е. его значение у всех создаваемых объектов одинаковое (аналог ключевого слова static в некоторых языках программирования).

Пояснительный текст в фигурных скобках может означать две различные конструкции. Если в этой строке имеется знак равенства, то вся запись Строка-свойство служит для указания дополнительных свойств атрибута, которые могут характеризовать особенности изменения значений атрибута в ходе выполнения программы. Фигурные скобки как раз и обозначают фиксированное значение соответствующего атрибута для класса в целом, которое должны принимать все вновь создаваемые экземпляры класса без исключения. Это значение принимается за исходное значение атрибута, которое не может быть переопределено в последующем.

Отсутствие строки-свойства по умолчанию трактуется так, что значение соответствующего атрибута может быть изменено в программе.

Знак "/" перед именем атрибута указывает на то, что данный атрибут является производным от некоторого другого атрибута этого же класса.

Производный атрибут (derived element) — атрибут класса, значение которого для отдельных объектов может быть вычислено посредством значений других атрибутов этого же объекта.

Операция (operation) - это сервис, предоставляемый каждым экземпляром или объектом класса по требованию своих клиентов, в качестве которых могут выступать другие объекты, в том числе и экземпляры данного класса.

Общий формат записи отдельной операции класса следующий:

```
<квантор видимости> <имя операции>(список параметров):
<выражение типа возвращаемого значения> {строка-свойство}
```

Квантор видимости операции аналогичен квантору видимости атрибутов.

Имя операции – обязательный элемент, должно начинаться со строчной (малой) буквы, и записываться без пробелов.

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

```
<направление параметра> <имя параметра> : <выражение типа> = <значение параметра по умолчанию>
```

Параметр (parameter) — спецификация переменной операции, которая может быть изменена, передана или возвращена.

Параметр может включать имя, тип, направление и значение по умолчанию. Направление параметра — есть одно из ключевых слов in, out или inout со значением in по умолчанию, в случае если вид параметра не указывается. Имя параметра есть идентификатор соответствующего формального параметра, при записи которого следуют правилам задания имен атрибутов.

Выражение типа возвращаемого значения указывает на тип данных значения, которое возвращается объектом после выполнения соответствующей операции. Две точки и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения. Для указания нескольких возвращаемых значений данный элемент спецификации операции может быть записан в виде списка отдельных выражений.

Операция с областью действия на весь класс показывается подчеркиванием имени и строки выражения типа. В этом случае под областью действия операции понимаются все объекты этого класса. В этом случае вся строка записи операции подчеркивается.

Строка-свойство служит для указания значений свойств, которые могут быть применены к данной операции. Строка-свойство может отсутствовать, если свойства не специфицированы.

Список формальных параметров и тип возвращаемого значения не обязателен.

В рамках профиля для процесса разработки ПО (The UML Profile for Software Development Processes) языка UML можно рассмотреть три специальных графических примитива, которые могут быть использованы для уточнения семантики отдельных классов при построении различных диаграмм:

Управляющий класс (control class) — класс, отвечающий за координацию действий других классов. На каждой диаграмме классов должен быть хотя бы один управляющий класс, причем количество посылаемых объектам управляющего класса сообщений мало, по сравнению с числом рассылаемых ими.

Управляющий класс отвечает за координацию действий других классов. Как правило, данный класс является активным и инициирует рассылку множества сообщений другим классам модели. Кроме специального обозначения управляющий класс может быть изображен в форме прямоугольника класса со стереотипом <<control>> (рис. 13, а).

Класс-сущность (entity class) — пассивный класс, информация о котором должна храниться постоянно и не уничтожаться с выключением системы. Класс-сущность содержит информацию, которая должна храниться постоянно и не уничтожается с уничтожением объектов данного класса или прекращением работы моделируемой системы, связанные с выключением системы или завершением программы.

Как правило, класс-сущность соответствует отдельной таблице базы данных. В этом случае его атрибуты являются полями таблицы, а операции – присоединенными или хранимыми процедурами. Этот класс пассивный и лишь принимает сообщения от других классов модели. Класс-сущность может быть изображен также стандартным образом в форме прямоугольника класса со стереотипом <<entity>> (рис. 13, б).

Граничный класс (boundary class) — класс, который располагается на границе

системы с внешней средой и непосредственно взаимодействует с актерами, но является составной частью системы. Граничный класс может быть изображен также стандартным образом в форме прямоугольника класса со стереотипом <<boundary>> (рис. 13, в).

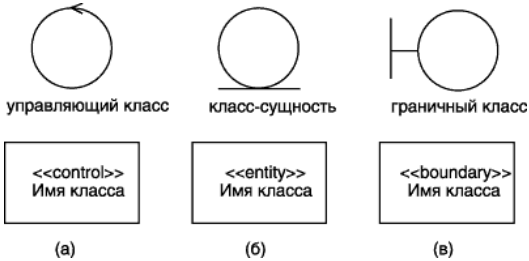


Рис. 13. Графическое изображение классов для моделирования ПО

В рамках профиля для бизнес-моделирования (The UML Profile for Business Modeling) языка UML модно рассмотреть три специальных графических примитива, которые могут быть использованы для уточнения семантики отдельных классов при построении моделей бизнес-систем:

Сотрудник (business worker) — класс, служащий на диаграмме классов для представления любого сотрудника, который является элементом бизнес-системы и взаимодействует с другими сотрудниками при реализации бизнес-процесса. Этот класс также может быть изображен в форме прямоугольника класса со стереотипом <<worker>> или <<internalWorker>> (рис. 14, а).

Сотрудник для связи с окружением (caseworker) – класс, служащий для представления в бизнес-системе такого сотрудника, который, являясь элементом бизнес-системы, непосредственно взаимодействует с актерами (бизнес-актерами) при реализации бизнес-процесса. Этот класс также может быть изображен в форме прямоугольника класса со стереотипом <<caseWorker>> (рис. 14, б).

Бизнес-сущность (business entity) — специальный случай класса-сущности, который также не инициирует никаких сообщений. Этот класс служит для сохранения информации о результатах выполнения бизнес-процесса в моделируемой бизнес-системе или организации. Этот класс также может быть изображен в форме прямоугольника класса со стереотипом <<business entity>> (рис. 14, в).

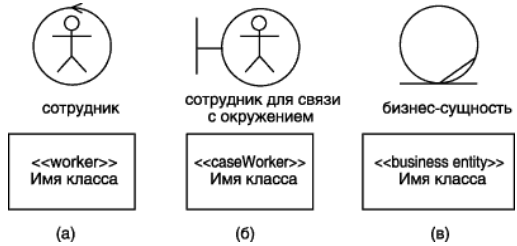


Рис. 14. Графическое изображение классов для моделирования бизнес-систем

Кроме внутреннего устройства классов важную роль при разработке проектиру-

емой системы имеют различные отношения между классами, которые также могут быть изображены на диаграмме классов. Базовые отношения, изображаемые на диаграммах классов:

- отношение ассоциации (association relationship);
- отношение обобщения (generalization relationship);
- отношение агрегации (aggregation relationship);
- отношение композиции (composition relationship).

Отношение ассоциации соответствует наличию произвольного отношения или взаимосвязи между классами. Оно обозначается сплошной линией со стрелкой или без нее и с дополнительными символами, которые характеризуют специальные свойства ассоциации.

Ассоциации применительно к диаграммам классов немного шире, чем применительно к диаграммам вариантов использования. В качестве дополнительных специальных символов могут использоваться имя ассоциации, символ навигации, а также имена и кратность классов-ролей ассоциации. Имя ассоциации - необязательный элемент ее обозначения. Однако если оно задано, то записывается с заглавной буквы рядом с линией ассоциации. Отдельные классы ассоциации играют определенную роль в соответствующем отношении, на что явно указывает имя концевых точек ассоциации на диаграмме. Наиболее простой случай данного отношения - *бинарная ассоциация*, которая служит для представления произвольного отношения между двумя классами. Она связывает в точности два различных класса и может быть ненаправленным (симметричным) или направленным отношением. Частный случай бинарной ассоциации - *рефлексивная_ассоциация*, которая связывает класс с самим собой. Ненаправленная бинарная ассоциация изображается линией без стрелки. Для нее на диаграмме может быть указан порядок чтения классов с использованием значка в форме треугольника рядом с именем данной ассоциации.

В качестве простого примера ненаправленной бинарной ассоциации можно рассмотреть отношение между двумя классами - классом «Компания» и классом «Сотрудник» (рис. 15). Они связаны между собой бинарной ассоциацией «Работает». Для данного отношения определен следующий порядок чтения следования классов - сотрудник работает в компании.

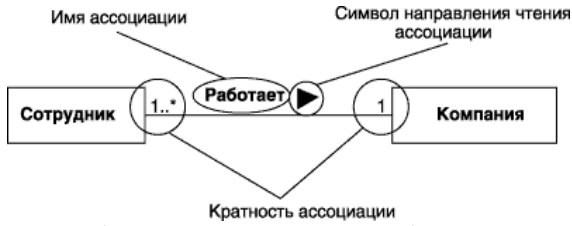


Рис. 15. Графическое изображение ненаправленной бинарной ассоциации между классами

Направленная бинарная ассоциация изображается сплошной линией с простой стрелкой на одной из ее концевых точек. Направление этой стрелки указывает на то,

какой класс является первым, а какой - вторым.

В качестве простого примера направленной бинарной ассоциации можно рассмотреть отношение между двумя классами - классом «Клиент» и классом «Счет» (рис. 16). Они связаны между собой бинарной ассоциацией с именем «Имеет», для которой определен порядок следования классов. Это означает, что конкретный объект класса «Клиент» всегда должен указываться первым при рассмотрении взаимосвязи с объектом класса «Счет», например, <клиент, счет_1, счет_2,…, счет_n>.

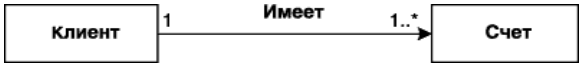


Рис. 16. Графическое изображение направленной бинарной ассоциации между классами

Частный случай отношения ассоциации - так называемая *исключающая_ассоциация* (Xor-association). Семантика данной ассоциации указывает на то, что из нескольких потенциально возможных вариантов данной ассоциации в каждый момент времени может использоваться только один.

На диаграмме классов исключающая ассоциация изображается пунктирной линией, соединяющей две и более ассоциации (рис. 6.7), рядом с которой записывается ограничение в форме строки текста в фигурных скобках: {xor}.

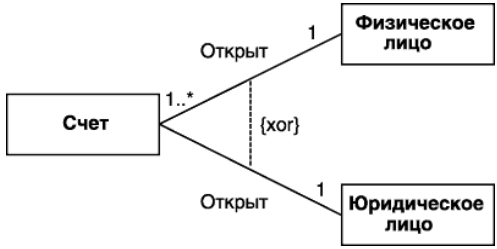


Рис. 17. Графическое изображение исключающей ассоциации между тремя классами

Тернарная ассоциация связывает отношением три класса. Ассоциация более высокой арности называется n -арной ассоциацией.

n-арная ассоциация - ассоциация между тремя и большим числом классов.

Каждый экземпляр такой ассоциации представляет собой упорядоченный набор (кортеж), содержащий n экземпляров из соответствующих классов. Такая ассоциация связывает отношением более чем три класса, при этом класс может участвовать в ассоциации более чем один раз.

Графически n -арная ассоциация обозначается ромбом, от которого ведут линии к символам классов данной ассоциации. Имя n -арной ассоциации записывается рядом с ромбом соответствующей ассоциации. Однако порядок классов в n -арной ассоциации, в отличие от порядка множеств в отношении, на диаграмме не фиксируется.

В качестве примера тернарной ассоциации можно рассмотреть отношение меж-

ду тремя классами: «Сотрудник», «Компания» и «Проект». Данная ассоциация указывает на наличие отношения между этими тремя классами, которое может представлять информацию о проектах, реализуемых в компании, и о сотрудниках, участвующих в выполнении отдельных проектов (рис. 18).

Рис. 18. Графическое изображение тернарной ассоциации между тремя классами

Класс может быть присоединен к линии ассоциации пунктирной линией, т.е. обеспечивать поддержку свойств соответствующей n-арной ассоциации, а сама n-арная ассоциация имеет атрибуты, операции и/или ассоциации. Такой класс является ассоциативным классом.

Класс ассоциация (association class) - модельный элемент, который одновременно является ассоциацией и классом.

Отдельный класс в ассоциации может играть определенную роль в данной ассоциации. Поэтому вводиться специальный элемент - *концевая точка ассоциации*, которая графически соответствует точке соединения линии ассоциации с отдельным классом.

Конец ассоциации (association end) - концевая точка ассоциации, которая связывает ассоциацию с классификатором. Конец ассоциации - часть ассоциации, но не класса. Каждая ассоциация может иметь два или больше концов ассоциации.

Роль (role) - имеющее имя специфическое поведение некоторой сущности, рассматриваемой в определенном контексте. Роль может быть статической или динамической.

Имя роли представляет собой строку текста рядом с концом ассоциации для соответствующего класса. Она указывает на специфическую роль, которую играет класс, являющийся концом рассматриваемой ассоциации. Имя роли не обязательный элемент обозначений.

Кратность_ассоциации относится к концам ассоциации и обозначается в виде интервала целых чисел, аналогично кратности атрибутов и операций классов, но без прямых скобок. Этот интервал записывается рядом с концом соответствующей ассоциации и означает потенциальное число отдельных экземпляров класса, которые могут иметь место, когда остальные экземпляры или объекты классов фиксированы.

Так, для примера (рис. 18) кратность " 1 " для класса «Компания» означает, что каждый сотрудник может работать только в одной компании. Кратность " 1..* " для класса «Сотрудник» означает, что в каждой компании могут работать несколько сотрудников, общее число которых заранее неизвестно и ничем не ограничено. Вместо кратности " 1..* " нельзя записать только символ " * ", поскольку последний означает кратность " 0..* ". Для данного примера это означало бы, что отдельные

компании могут совсем не иметь сотрудников в своем штате. Такая кратность приемлема в ситуациях, когда в компании вообще не выполняется никаких проектов.

Имя ассоциации рассматривается в качестве отдельного атрибута у соответствующих классов ассоциаций и может быть указано на диаграмме явным способом в определенной секции прямоугольника класса.

Ассоциация является наиболее общей формой отношения в языке UML. Все другие типы рассматриваемых отношений можно считать частным случаем данного отношения.

Отношение обобщения является отношением классификации между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком).

Менее общий элемент модели должен быть согласован с более общим элементом и может содержать дополнительную информацию. Отношение обобщения описывает иерархическое строение классов и наследование их свойств и поведения.

Согласно одному из главных принципов методологии ООАП - наследованию, класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет собственные свойства и поведение, которые могут отсутствовать у класса-предка.

Отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рис. 19). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее начало - на более специальный класс (класс-потомок или подкласс).

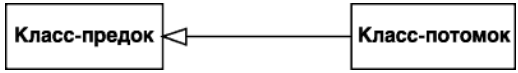


Рис. 19. Графическое изображение отношения обобщения в языке UML

От одного класса-предка одновременно могут наследовать несколько классов-потомков. т.е. в класс-предок входит несколько линий со стрелками.

Пример: класс «Транспортное средство» (курсив обозначает абстрактный класс) может выступать в качестве суперкласса для подклассов, соответствующих конкретным транспортным средствам, таким как: «Автомобиль», «Автобус», «Трактор» и другим (рис 20).

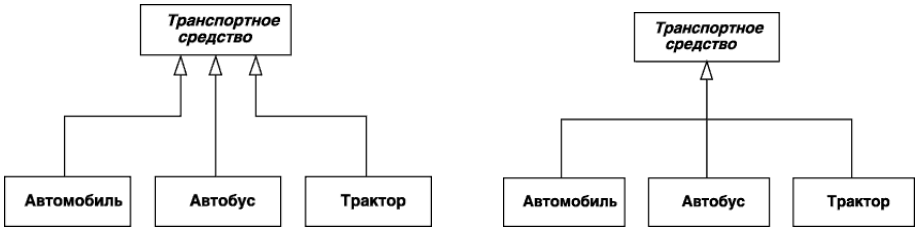


Рис. 20. а) Пример графического изображения отношения обобщения для нескольких классов-потомков; б) Альтернативный вариант графического изображения от-

ношения обобщения классов для случая объединения отдельных линий

Графическое изображение отношения обобщения по форме соответствует иерархическому дереву, где класс-предок является корнем дерева, а классы-потомки - его листьями.

В дополнение к простой стрелке обобщения может быть присоединена строка текста, указывающая на специальные свойства этого отношения в форме ограничения. Этот текст будет относиться ко всем линиям обобщения, которые идут к классам-потомкам. Поскольку отмеченное свойство касается всех подклассов данного отношения, именно поэтому спецификация этого свойства осуществляется в форме ограничения, которое должно быть записано в фигурных скобках.

В качестве ограничений могут быть использованы следующие ключевые слова языка UML:

- {complete} - означает, что в данном отношении обобщения специфицированы все классы-потомки, и других классов-потомков у данного класса-предка быть не может.
- {incomplete} - означает случай, противоположный первому. А именно, предполагается, что на диаграмме указаны не все классы-потомки. В последующем возможно разработчик восполнит их перечень, не изменяя уже построенную диаграмму.
- {disjoint} - означает, что классы-потомки не могут содержать объектов, одновременно являющихся экземплярами двух или более классов.
- {overlapping} - случай, противоположный предыдущему. А именно, предполагается, что отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам.

С учетом дополнительного использования стандартного ограничения диаграмма классов (рис. 20) может быть уточнена (рис. 21).

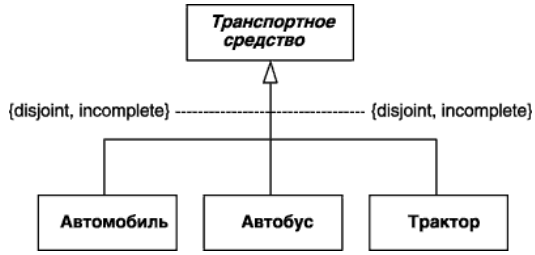


Рис. 21. Вариант уточненного графического изображения отношения обобщения классов с использованием строки-ограничения

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой сущность, которая включает в себя в качестве составных частей другие сущности. Данное отношение применяется для представления системных взаимосвязей типа "часть-целое" и показывает, из каких элементов состоит система, и как они связаны между собой.

Очевидно, что рассматриваемое в таком аспекте деление системы на составные части представляет собой иерархию, но принципиально отличную от той, которая порождается отношением обобщения. Отличие заключается в том, что части системы никак не обязаны наследовать ее свойства и поведение, поскольку являются самостоятельными сущностями. Более того, части целого обладают собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого. Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой не закрашенный внутри ромб. Этот ромб указывает на тот класс, который представляет собой "целое" или класс-контейнер. Остальные классы являются его "частями" (рис. 22).

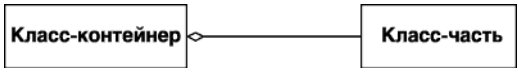


Рис. 22. Графическое изображение отношения агрегации в языке UML

В качестве примера отношения агрегации можно рассмотреть взаимосвязь типа "часть-целое", которая имеет место между классом «Системный блок» персонального компьютера и его составными частями: «Процессор», «Материнская плата», «Оперативная память», «Жесткий диск» и «Дисковод гибких дисков».

Рис. 23. Диаграмма классов для иллюстрации отношения агрегации на примере системного блока ПК

Отношение композиции - частный случай отношения агрегации, и служит для спецификации более сильной формы отношения "часть-целое", при которой составляющие части тесно взаимосвязаны с целым. Особенность: с уничтожением целого уничтожаются и все его составные части.

Композит (composite) - класс, который связан отношением композиции с одним или большим числом классов. Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот класс, который представляет собой класс-композит. Остальные классы являются его "частями" (рис. 24).

Рис. 24.

Для отношений композиции и агрегации могут использоваться дополнительные обозначения, применяемые для отношения ассоциации. А именно, могут указы-

ваться кратности отдельных классов, которые в общем случае не обязательны. Применительно к описанному выше примеру класс Окно программы является классом-композитом, а взаимосвязи составляющих его частей могут быть изображены следующей диаграммой классов (рис. 25).

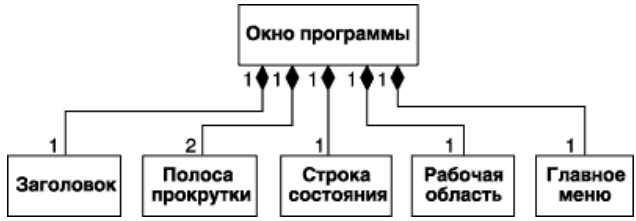


Рис. 25. Диаграмма классов для иллюстрации отношения композиции на примере класса-композита «Окно программы»

Процесс разработки диаграммы классов занимает центральное место при разработке проектов сложных систем. От умения правильно выбрать классы и установить между ними взаимосвязи часто зависит не только успех процесса проектирования, но и производительность выполнения программы. Как показывает практика ООАП, каждый программист в той или иной степени использует личный опыт при разработке новых проектов. Это обусловлено желанием свести новую задачу к уже решенным, чтобы иметь возможность применять не только проверенные фрагменты программного кода, но и отдельные компоненты или библиотеки компонентов.

При определении классов, атрибутов и операций, задании их имен и типов перед разработчиком всегда встает вопрос: какой язык взять за основу.

После разработки диаграммы классов процесс ООАП может быть продолжен в двух направлениях. С одной стороны, если поведение системы тривиально, то можно приступить к разработке диаграмм кооперации и последовательности. Особенности построения диаграмм кооперации будут рассмотрены в следующей главе. Однако для сложных динамических систем, в частности для параллельных систем и систем реального времени, важнейшим аспектом их функционирования является поведение. Специфика поведения таких систем может быть представлена на диаграммах состояний и деятельности, разработка которых в общем случае не обязательна, а определяется конкретным проектом.

Пример: Программное средство представляет собой базу данных «Автоматизация процесса составления расписания в учебном заведении». Программное средство обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствие с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

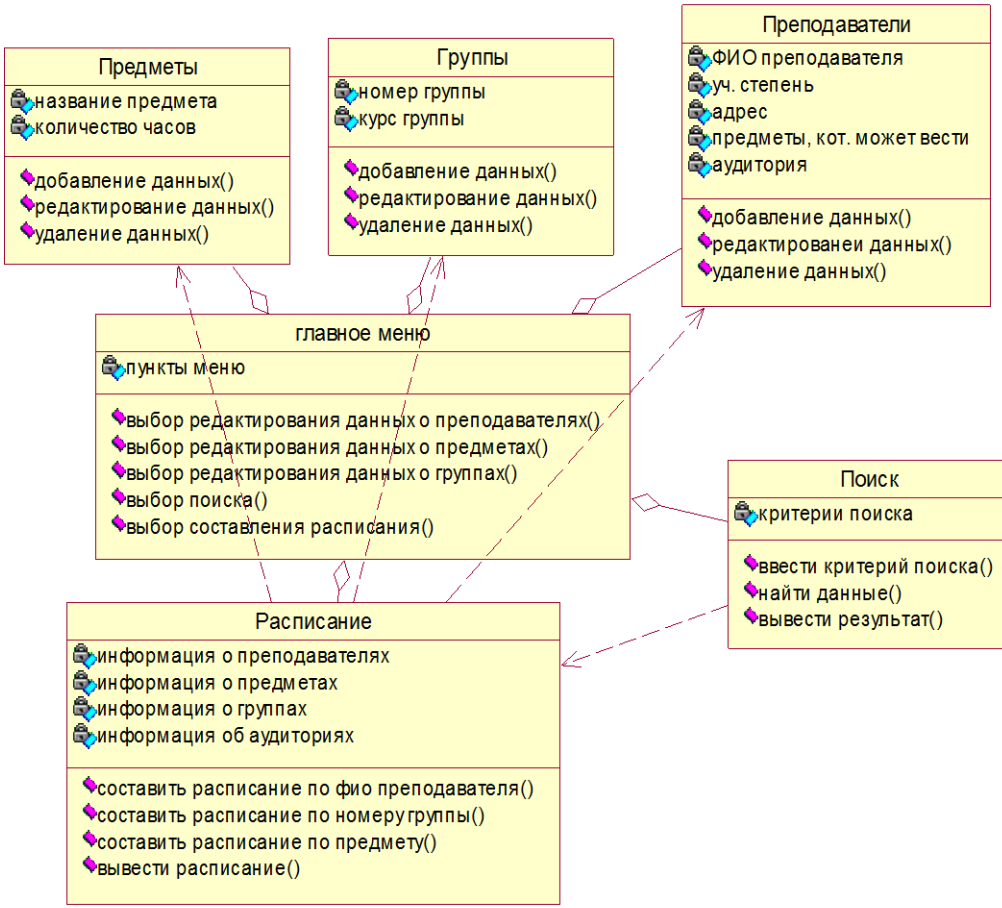


Рис. 26. Пример диаграммы классов

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Диаграмма классов”.
2. Разработать диаграмму классов для произвольной системы индивидуального задания (Приложение А).
3. Оформить отчет, включив в него описание всех компонентов диаграммы классов согласно индивидуальному варианту задания, ответы на контрольные вопросы.

Контрольные вопросы

1. Дайте определение понятию «диаграмма классов» и ее назначение.
2. Дайте определение следующим понятиям «класс», «атрибут», «операция», «отношение», «интерфейс».
3. Опишите отношение ассоциаций между экземплярами классов.
4. Опишите отношение обобщения между экземплярами классов.
5. Опишите отношение агрегации между экземплярами классов.
6. Опишите отношение композиции между экземплярами классов.
7. Приведите пример графического представления основных компонентов диаграммы классов.
8. В чем различие конкретного класса от абстрактного?
9. Дайте определение «квантор видимости» и его классификацию.
10. Какие дополнительные графические примитивы существуют в графической нотации языка UML диаграмме классов?

ЛАБОРАТОРНАЯ РАБОТА № 3
ДИАГРАММА КООПЕРАЦИИ

Цель работы: закрепление теоретических сведений о диаграмме кооперации; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме кооперации.

Краткие теоретические сведения

Диаграмма кооперации (collaboration diagram) предназначена для описания поведения системы на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели или реализовать некоторый вариант использования.

Кооперация (collaboration) — спецификация множества объектов отдельных классов, совместно взаимодействующих с целью реализации отдельных вариантов использования в общем контексте моделируемой системы.

Кооперация определяет структуру поведения системы в терминах взаимодействия участников этой кооперации.

На диаграмме кооперации размещаются объекты, представляющие собой экземпляры классов, связи между ними, которые в свою очередь являются экземплярами ассоциаций, и сообщения. Связи дополняются стрелками сообщений, а также именами ролей, которые играют объекты в данной взаимосвязи. На диаграмме кооперации показываются структурные отношения между объектами в виде различных соединительных линий и изображаются динамические взаимосвязи — потоки сообщений в форме стрелок с указанием направления рядом с соединительными линиями между объектами, при этом задаются имена сообщений и их порядковые номера в общей последовательности сообщений.

Одна и та же совокупность объектов может участвовать в реализации различных

кооперацій. В зависимости от рассматриваемой кооперации, могут изменяться как связи между отдельными объектами, так и поток сообщений между ними. Именно это отличает диаграмму кооперации от диаграммы классов, на которой должны быть указаны все без исключения классы, их атрибуты и операции, а также все ассоциации и другие структурные отношения между элементами модели.

Объект(object) — сущность с хорошо определенными границами и индивидуальностью, которая инкапсулирует состояние и поведение. Объект создается на этапе реализации модели или выполнения программы. Он имеет собственное имя и конкретные значения атрибутов.

Для диаграмм кооперации имя объекта – строка текста, разделенная двоеточием: `<собственное имя объекта>'/'<Имя роли класса>:<Имя класса>.`

Имя роли класса указывается в том случае, когда соответствующий класс отсутствует в модели. Имя класса – это имя одного из классов, представленного на диаграмме классов.

Если указано собственное имя объекта, то оно должно начинаться со строчной буквы. Имя объекта, имя роли с символом "/" или имя класса могут отсутствовать, но ":" всегда должно стоять перед именем класса, а "/" – перед именем роли.

Следующие варианты возможных записей полного имени объекта:

- o : C – объект с собственным именем o, экземпляр класса C.
- : C – анонимный объект, экземпляр класса C.
- o : (или o) – объект-сирота с собственным именем o.
- o / R : C – объект с собственным именем o, экземпляр класса C, играющий роль R.
- / R : C – анонимный объект, экземпляр класса C, играющий роль R.
- o / R – объект-сирота с собственным именем o, играющий роль R.
- / R –анонимный объект и одновременно объект-сирота, играющий роль R.

Рис. 27. Примеры графических изображений объектов на диаграммах кооперации

Если собственное имя объекта отсутствует (ставится двоеточие перед именем соответствующего класса (рис. 27, в), то такой объект принято называть *анонимным*.

Если отсутствует имя класса – такой объект называется *сиротой* (записывается только собственное имя объекта, двоеточие не ставится, имя класса не указываются (рис. 27, г). Для отдельных объектов (рис. 27, д, е) могут быть дополнительно указаны роли, которые они играют в кооперации.

Все объекты делятся на две категории: пассивные и активные. *Пассивный* объект оперирует только данными и не может инициировать деятельность по управлению другими объектами. Однако пассивные объекты могут посылать сигналы в процессе выполнения запросов, которые они обрабатывают. На диаграмме кооперации пассивные объекты изображаются обычным образом без дополнительных стереотипов.

Активный объект имеет собственный процесс управления и может инициировать деятельность по управлению другими объектами.

Активный объект на диаграмме кооперации обозначается прямоугольником с утолщенными границами (рис. 28). Каждый активный объект является владельцем определенного процесса управления.

Пример: активный объект «a : Клиент» является инициатором открытия счета, который представлен анонимным объектом «: Счет».

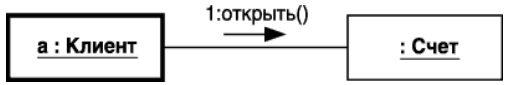


Рис. 28. Графическое изображение активного объекта (слева) на диаграмме кооперации

Мультиобъект(multiobject) представляет собой множество анонимных объектов, которые могут быть образованы на основе одного класса. На диаграмме кооперации мультиобъект используется для того, чтобы показать операции и сигналы, которые адресованы всему множеству анонимных объектов. Мультиобъект изображается двумя прямоугольниками, один из которых выступает из-за верхней правой вершины другого (рис. 29, а). При этом стрелка взаимосвязи относится ко всему множеству объектов, которые обозначает данный мультиобъект. На диаграмме кооперации может быть явно указано отношение агрегации (композиции) между мультиобъектом и отдельным объектом из его множества (рис. 29, б).

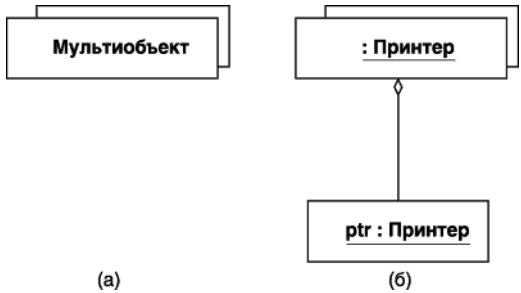


Рис. 29. Графическое изображение мультиобъектов на диаграмме кооперации

Пример: ситуация с отправкой почтового сообщения клиенту из редактора электронной почты (рис. 30). Анонимный активный объект класса «: РедакторEmail» вначале посылает сообщение анонимному мультиобъекту класса «Клиент». Это сообщение инициирует выбор единственного объекта класса «Клиент». После этого выбранному объекту посылается сообщение о необходимости отправить электронное письмо.

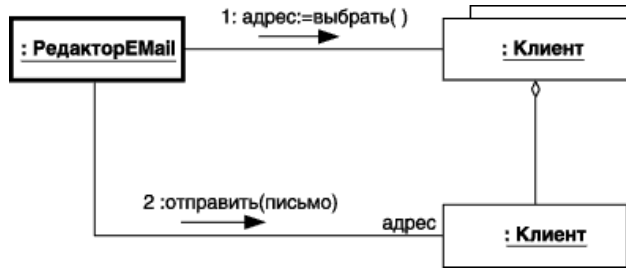


Рис. 30. Фрагмент диаграммы кооперации для выбора адреса клиента для отправки электронного письма

Составной объект (composite object) или объект-композит предназначен для представления объекта, имеющего собственную структуру и внутренние потоки управления.

Составной объект является экземпляром класса-композита, который связан отношением композиции со своими частями. На диаграммах кооперации составной объект изображается как обычный объект, состоящий из двух секций: верхней и нижней. В верхней секции записывается имя составного объекта, а в нижней – его объекты-части вместо списка атрибутов (рис. 31).

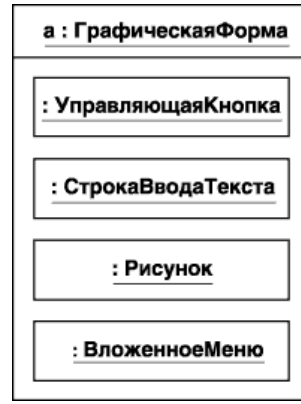


Рис. 31. Графическое изображение составного объекта на диаграмме кооперации

При изображении диаграммы кооперации отношения между объектами описываются с помощью связей, которые являются экземплярами соответствующих ассо-

циаций.

При изображении диаграммы кооперации отношения между объектами описываются с помощью связей, которые являются экземплярами соответствующих ассоциаций.

Связь(link) — любое семантическое отношение между некоторой совокупностью объектов.

Бинарная связь изображается отрезком сплошной линии, соединяющей два прямоугольника объектов, на концах линии можно указать имена ролей.

Связи на диаграмме кооперации могут быть только анонимными и при необходимости записываются без двоеточия перед именем ассоциации. Имена связей и кратность концевых точек, как правило, на диаграммах кооперации не указываются. Обозначения агрегации и композиции могут присутствовать на отдельных концах связей.

Пример: обобщенная схема компании с именем «с», которая состоит из департаментов (анонимный мультиобъект класса «Департамент»). В последние входят «Сотрудники». Рефлексивная связь указывает на то, что руководитель департамента является одновременно и его сотрудником.

Рис. 32 Графическое изображение связей с различными стереотипами

Связь может иметь некоторые стереотипы:

- «association» – ассоциация (предполагается по умолчанию, поэтому может не указываться);
- «parameter» – соответствующий объект может быть только параметром метода;
- «local» – область видимости переменной ограничена только соседним объектом;
- «global» – область видимости переменной распространяется на всю диаграмму кооперации;
- «self» – рефлексивная связь объекта с самим собой, которая допускает передачу объектом сообщения самому себе.

Каждое взаимодействие описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой.

Сообщение (message) — спецификация передачи информации от одного элемента модели к другому с ожиданием выполнения определенных действий со стороны принимающего элемента.

При этом первый объект предполагает, что после получения сообщения вторым объектом последует выполнение некоторого действия. На диаграмме кооперации сообщение является причиной или стимулом начала выполнения операций, отправки сигналов, создания и уничтожения отдельных объектов. Связь обеспечивает канал для направленной передачи сообщений между объектами от объекта-источника к объекту-получателю.

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

Сообщения в языке UML специфицируют роли, которые играют объекты – отправитель и получатель сообщения. Сообщения на диаграмме кооперации изображаются дополнительными стрелками рядом с соответствующей связью или ролью ассоциации. Направление стрелки указывает на получателя сообщения. На диаграммах кооперации может использоваться один из трех типов стрелок для обозначения сообщений (рис. 33).

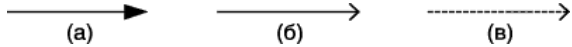


Рис. 33. Графическое изображение различных типов сообщений на диаграмме кооперации

Сплошная линия с треугольной стрелкой (рис. 33, а) обозначает вызов процедуры (операции) или передачу потока управления. Сообщения этого типа могут быть использованы параллельно активными объектами, когда один из них передает сообщение этого типа и ожидает, пока не закончится некоторая последовательность действий, выполняемая вторым объектом. Обычно все такие сообщения синхронны, т. е. инициируются по завершении деятельности или при выполнении определенного условия.

Сплошная линия с V-образной стрелкой (рис. 33, б) обозначает асинхронное сообщение в простом потоке управления. В этом случае клиент передает асинхронное сообщение и продолжает выполнять свою деятельность, не ожидая ответа от сервера.

Пунктирная линия с V-образной стрелкой (рис. 33, в) обозначает возврат из вызова процедуры. Стрелки этого типа зачастую отсутствуют на диаграммах кооперации, поскольку неявно предполагается их существование после окончания процесса выполнения операции или деятельности.

Каждое сообщение может быть помечено строкой текста, которая имеет следующий формат:

<Предшествующие сообщения> <Выражение последовательности> <Возвращаемое значение := имя сообщения> (Список аргументов)

Предшествующие сообщения — это разделенные запятыми номера сообщений, записанные перед наклонной чертой: <Номер сообщения','>* <'/'>. Если список номеров сообщений пуст, то вся запись, включая наклонную черту, опускается. Если номера сообщений указываются, то они должны соответствовать номерам других сообщений на этой же диаграмме кооперации. Смысл указания предшествующих сообщений заключается в том, что данное сообщение не может быть передано, пока не будут переданы своим адресатам все сообщения, номера которых записаны в данном списке.

Выражение последовательности — это разделенный точками список отдельных термов последовательностей, после которого записывается двоеточие: <Терм последовательности'.'…> ':'

Каждый из термов представляет отдельный уровень процедурной вложенности в форме законченной итерации. Наиболее верхний уровень соответствует самому левому терму последовательности. Если все потоки управления параллельные, то вложенность отсутствует. Каждый терм последовательности имеет следующий синтаксис: [Целое число | Имя] [Рекуррентность].

Целое число указывает на порядковый номер сообщения в процедурной последовательности верхнего уровня. Сообщения, номера которых отличаются на единицу, следуют подряд один за другим.

Имя в форме буквы алфавита используется для спецификации параллельных потоков (нитей) управления. Сообщения, которые отличаются только именем, являются параллельными на этом уровне вложенности. На одном уровне вложенности все нити управления эквивалентны в смысле приоритета передачи сообщений.

Рекуррентность используется для указания итеративного или условного характера выполнения передачи сообщений. Семантика рекуррентности представляет ноль или больше сообщений, которые должны быть выполнены в зависимости от записанного условия. Возможны два варианта записи рекуррентности:

- '*"['Предложение-итерация']' для записи итеративного выполнения соответствующего выражения. Итерация представляет последовательность сообщений одного уровня вложенности. Предложение-итерация может быть опущено, если количество итераций никак не специфицируется. Наиболее часто предложение-итерация записывается на псевдокоде или языке программирования. В языке UML формат записи этого предложения строгим образом не определен.
- '['Предложение-условие']' для записи ветвления. Эта форма записи специфицирует условие для данного сообщения, передача которого по данной ветви возможна только при его истинности.

В общем случае предложение-условие - обычное булевское выражение и предназначено для синхронизации отдельных нитей потока управления. Записывается в квадратных скобках и может быть опущено, если оно отсутствует у данного сообщения. Наличие этого условия обеспечивает передачу сообщения только в том случае, если это условие принимает значение "истина". Предложение-условие может быть записано на обычном тексте, псевдокоде или некотором языке программирования.

Предложение-условие записывается так же, как и итерация, но без звездочки. Это можно понимать как некоторую одношаговую итерацию. В общем случае предполагается, что специфицированная итерация выполняется последовательно. Если необходимо отметить возможность параллельного выполнения итерации, то для этой цели в языке UML используется символ "*||". Итерация не распространяется на вложенные уровни данного потока или нити. Каждый уровень должен иметь собственное представление для итеративного повторения процедурной последовательности.

Имя сообщения, записанное в сигнатуре после возвращаемого значения, означает имя события, которое инициируется объектом-получателем сообщения после его приема. Наиболее часто таким событием является вызов операции у объекта-получателя. Это может быть реализовано различными способами, один из которых – явное указание в качестве имени сообщения вызываемой операции. Тогда соответствующая операция должна быть определена в том классе, которому принадлежит объект-получатель.

Список аргументов представляет собой разделенные запятыми и заключенные в круглые скобки действительные параметры той операции, вызов которой инициируется данным сообщением. Список аргументов может быть пустым, однако скобки все равно записываются. Для записи аргументов также может быть использован некоторый псевдокод или язык программирования.

В языке UML предусмотрены стандартные действия, выполняемые в ответ на получение соответствующего сообщения. Они могут быть явно указаны на диаграмме кооперации в форме стереотипа перед именем сообщения, к которому они относятся, или выше его.

В языке UML определены следующие стереотипы сообщений:

- «call» (вызвать) – сообщение, требующее вызова операции или процедуры объекта-получателя. Если сообщение с этим стереотипом рефлексивное, то оно инициирует локальный вызов операции у пославшего это сообщение объекта.
- «return» (возвратить) – сообщение, возвращающее значение выполненной операции или процедуры вызвавшему ее объекту. Значение результата может инициировать ветвление потока управления.
- «create» (создать) – сообщение, требующее создания другого объекта для выполнения определенных действий. Созданный объект может стать активным (ему передается поток управления), а может остаться пассивным.
- «destroy» (уничтожить) – сообщение с явным требованием уничтожить соответствующий объект. Посылается в том случае, когда необходимо прекратить нежелательные действия со стороны существующего в системе объекта, либо когда объект больше не нужен и должен освободить задействованные им системные ресурсы.
- «send» (послать) – обозначает посылку другому объекту сигнала, который асинхронно инициируется одним объектом и принимается (перехватывается) другим. Отличие сигнала от сообщения заключается в том, что сигнал должен быть явно описан в том классе, объект которого инициирует его передачу.

Построение диаграммы кооперации можно начинать сразу после построения диаграммы классов. При разработке диаграмм кооперации вначале изображаются

объекты и связи между ними. Далее на диаграмму кооперации необходимо нанести все сообщения, указав их порядок и другие семантические особенности. Эта диаграмма может содержать только те объекты и связи, которые уже определены на построенной ранее диаграмме классов.

Процесс построения диаграммы кооперации должен быть согласован с процессами построения диаграммы классов и диаграммы последовательности. В первом случае, как уже отмечалось, необходимо следить за использованием только тех объектов, для которых определены порождающие их классы. Во втором случае необходимо согласовывать последовательности передаваемых сообщений. Речь идет о том, что не допускается различный порядок следования сообщений для моделирования одного и того же взаимодействия на диаграмме кооперации и диаграмме последовательности.

Диаграмма кооперации, с одной стороны, обеспечивает концептуально согласованный переход от статической модели диаграммы классов к динамическим моделям поведения, представляемым диаграммами последовательности, состояний и деятельности. С другой стороны, диаграмма этого типа предопределяет особенности реализации модели на диаграммах компонентов и развертывания.

Пример: Программное средство представляет среду для формирования отчетов по лекционному материалу. Пользователь может вводить свою информацию в отчеты, изменять параметры. После оформления отчетов пользователю предоставлена возможность сохранение отчета.

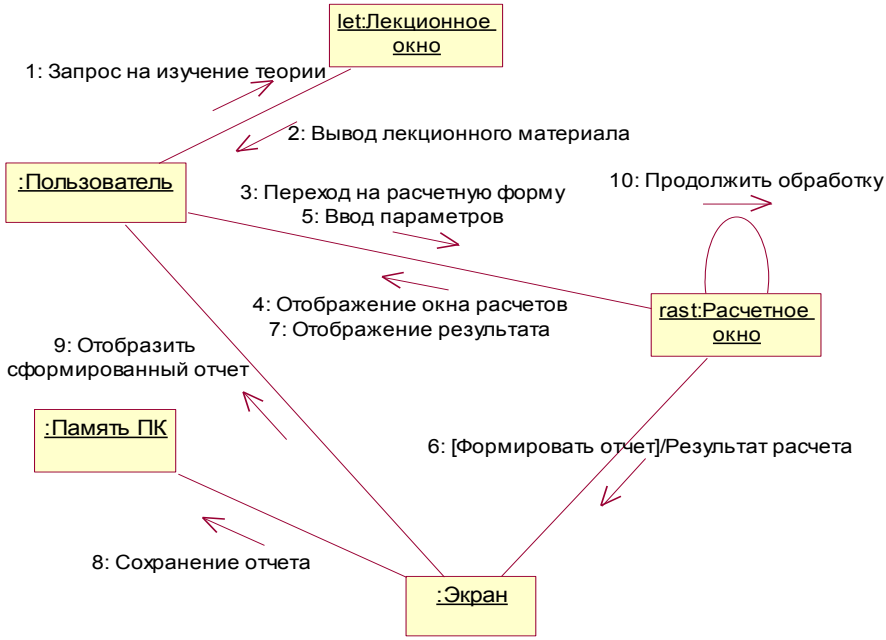


Рис. 34. Пример диаграммы кооперации

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Диаграмма кооперации”.
2. Разработать диаграмму кооперации для произвольной системы индивидуального задания (Приложение А).
3. Оформить отчет, включив в него описание всех компонентов диаграммы кооперации согласно индивидуальному варианту задания, ответы на контрольные вопросы.

Контрольные вопросы

1. Дайте определение понятию «диаграмма кооперации».
2. Опишите назначение диаграммы кооперации.
3. Что из себя представляет объект в диаграмме кооперации? Как полностью записывается имя объекта? Назовите варианты записи объекта.
4. На какие две категории делятся объекты?
5. Дайте определение понятиям «мультиобъект», «составной объект». Каково их графическое изображение?
6. Что понимается под связью в диаграмме кооперации? Какие типы связи возможны между объектами на диаграмме кооперации?
7. Дайте определение понятию «сообщение» в диаграмме кооперации и назовите его формат записи. Графическое изображение сообщений.
8. Какие стереотипы сообщений существуют в языке UML? Охарактеризуйте каждый из них.

ЛАБОРАТОРНАЯ РАБОТА № 4
ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ

Цель работы: закрепление теоретических сведений о диаграмме последовательности; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме последовательности.

Краткие теоретические сведения

Диаграмма_последовательности (sequence diagram) – диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления.

На диаграмме последовательности неявно присутствует ось времени, что позволяет визуализировать временные отношения между передаваемыми сообщениями. С помощью диаграммы последовательности можно представить взаимодействие элементов модели как своеобразный временной график "жизни" всей совокупности объектов, связанных между собой для реализации варианта использования программной системы, достижения цели или выполнения задачи.

На диаграмме последовательности изображаются объекты, которые непосредственно участвуют во взаимодействии, при этом никакие статические связи с дру-

гими объектами не визуализируются. Диаграмма последовательности имеет как бы два измерения: одно - слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта; второе – вертикальная временная ось, направленная сверху вниз.

Каждый объект графически изображается в форме прямоугольника и располагается в верхней части своей линии жизни (рис. 35). Внутри прямоугольника записываются собственное имя объекта и имя класса, разделенные двоеточием, запись подчеркивается, что является признаком объекта, который, как указывалось ранее, представляет собой экземпляр класса.

Объекты записываются так же, как и в диаграмме кооперации. **Анонимный объект** – такой объект, для которого отсутствует собственное имя, но указано имя класса. **Объект-сирота** – такой объект, для которого может отсутствовать имя класса, но указано собственное имя объекта. Роль классов в именах объектов на диаграммах последовательности, как правило, не указывается.

Крайним слева на диаграмме изображается объект - инициатор моделируемого процесса взаимодействия (объект a на рис. 35). Правее - другой объект, который непосредственно взаимодействует с первым, т.е. порядок расположения объектов на диаграмме последовательности определяется исключительно соображениями удобства визуализации их взаимодействия друг с другом.

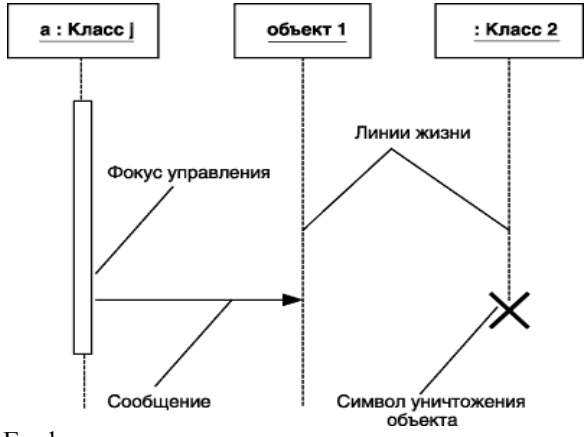


Рис. 35. Графические элементы диаграммы последовательности

Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом процесс взаимодействия объектов реализуется посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и образуют определенный порядок относительно времени своей инициализации. Другими словами, сообщения, расположенные на диаграмме последовательности выше, передаются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа "раньше-позже".

Линия жизни объекта (object lifeline) – вертикальная линия на диаграмме последовательности, которая представляет существование объекта в течение определенного периода времени. Линия жизни объекта изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей рабочей области диаграммы последовательности от самой верхней ее части до самой нижней («объект 1» и анонимный объект «Класса 2»).

Отдельные объекты, закончив выполнение своих операций, 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 применяется специальный символ в форме латинской буквы "X". На рис. 36 этот символ используется для уничтожения анонимного объекта, образованного от «Класса 3» . Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

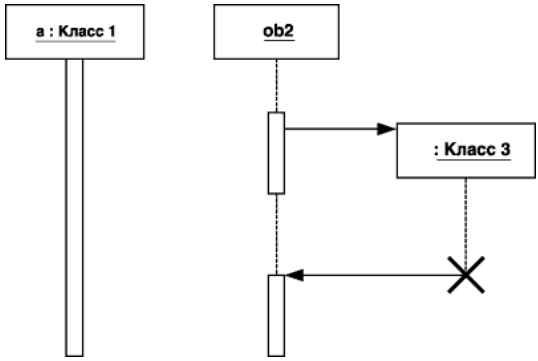


Рис. 36. Графическое изображение линий жизни и фокусов управления объектов

Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее производительность. В этом случае прямоугольник такого объекта изображается не в верхней части диаграммы последовательности, а в той, которая соответствует моменту создания объекта (анонимный объект, образованный от «Класса 3» на рис. 36). При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Объект создается со своей линией жизни а, возможно, и с фокусом управления.

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия, или в состоянии пассивного ожидания сообщений от других объектов. Фокус управления - символ, применяемый для того, чтобы явно выделить

подобную активность объектов на диаграммах последовательности.

Фокус_управления (focus of control) - специальный символ на диаграмме последовательности, указывающий период времени, в течение которого объект выполняет некоторое действие, находясь в активном состоянии.

Фокус управления изображается в форме вытянутого узкого прямоугольника («объект а» на рис. 35), верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона - окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни («объект а» на рис. 36), если на всем ее протяжении он активен.

Периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта фокусы управления изменяют свое изображение на линию жизни и наоборот (объект сирота «ob2» на рис. 36). Получить фокус управления может только объект, у которого в этот момент имеется линия жизни. Если же объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него может быть создан лишь экземпляр этого же класса, который, строго говоря, будет другим объектом.

В отдельных случаях объект может посылать сообщения самому себе, инициируя так называемые *рефлексивные* сообщения. Для этой цели служит специальное изображение (сообщение у «объекта а» на рис. 37). Такие сообщения изображаются в форме сообщения, начало и конец которого соприкасаются с линией жизни или фокусом управления одного и того же объекта. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента.

Если в результате рефлексивного сообщения создается новый подпроцесс или нить управления, то говорят о рекурсивном или вложенном фокусе управления. На диаграмме последовательности рекурсия обозначается небольшим прямоугольником, присоединенным к правой стороне фокуса управления того объекта, для которого изображается данное рекурсивное взаимодействие (анонимный объект «Класса 2» на рис. 37).

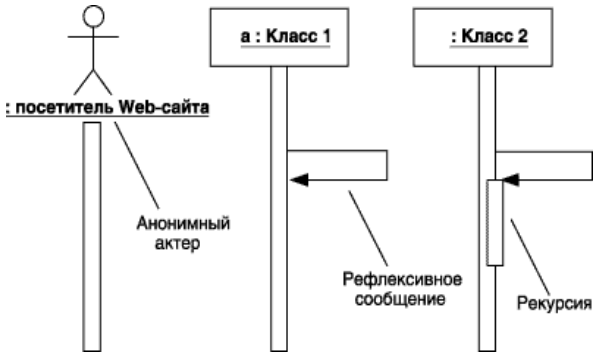


Рис. 37. Графическое изображение актера, рефлексивного сообщения и рекурсии

Сообщения на диаграмме последовательности аналогичны сообщениям на диа-

грамме кооперации, но имеют дополнительные семантические особенности. На диаграмме последовательности все сообщения упорядочены по времени своей передачи в моделируемой системе.

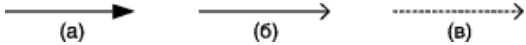


Рис. 38. Графическое изображение различных видов сообщений между объектами на диаграмме последовательности

Первая разновидность сообщения (рис. 38, а) наиболее распространена и используется для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления. Начало этой стрелки, как правило, соприкасается с фокусом управления того объекта-клиента, который инициирует это сообщение. Конец стрелки соприкасается с линией жизни того объекта, который принимает это сообщение и выполняет в ответ определенные действия. При этом принимающий объект может получить фокус управления, становясь в этом случае активным. Передающий объект может потерять фокус управления или остаться активным.

Вторая разновидность сообщения (рис. 38, б) используется для обозначения простого асинхронного сообщения, которое передается в произвольный момент времени. Передача такого сообщения обычно не сопровождается получением фокуса управления объектом-получателем.

Третья разновидность сообщения (рис. 38, в) используется для возврата из вызова процедуры. Примером может служить простое сообщение о завершении вычислений без предоставления результата расчетов объекту-клиенту. В процедурных потоках управления эта стрелка может быть опущена, поскольку ее наличие неявно предполагается в конце активизации объекта. В то же время считается, что каждый вызов процедуры имеет свою пару - возврат вызова. Для непроцедурных потоков управления, включая параллельные и асинхронные сообщения, стрелка возврата должна указываться явным образом.

Обычно сообщения изображаются горизонтальными стрелками, соединяющими линии жизни или фокусы управления двух объектов на диаграмме последовательности. При этом неявно предполагается, что время передачи сообщения достаточно мало по сравнению с процессами выполнения действий объектами. Считается также, что за время передачи сообщения с соответствующими объектами не может произойти никаких событий. Другими словами, состояния объектов не изменяются. Если же это предположение не может быть признано справедливым, то стрелка сообщения изображается под наклоном, так чтобы конец стрелки располагался ниже ее начала.

Каждое сообщение на диаграмме последовательности ассоциируется с определенной операцией, которая должна быть выполнена принявшим его объектом. При этом операция может иметь аргументы или параметры, значения которых влияют на получение различных результатов. Соответствующие параметры операции будет иметь и вызывающее это действие сообщение. Более того, значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвление или альтернативные пути основного потока управления.

Одна из особенностей диаграммы последовательности - возможность визуализировать простое *ветвление процесса*. Для изображения ветвления используются две или более стрелки, выходящие из одной точки фокуса управления объекта (объект «ob1» на рис. 39). При этом рядом с каждой из них должно быть явно указано соответствующее условие ветви в форме булевского выражения.

Количество ветвей может быть произвольным, но наличие ветвлений может существенно усложнить интерпретацию диаграммы последовательности. Предложение-условие должно быть явно указано для каждой ветви и записывается в форме обычного текста, псевдокода или выражения языка программирования. Это выражение всегда должно возвращать некоторое булевское выражение. Запись этих условий должна исключать одновременную передачу альтернативных сообщений по двум и более ветвям. В противном случае на диаграмме последовательности может возникнуть конфликт ветвления.

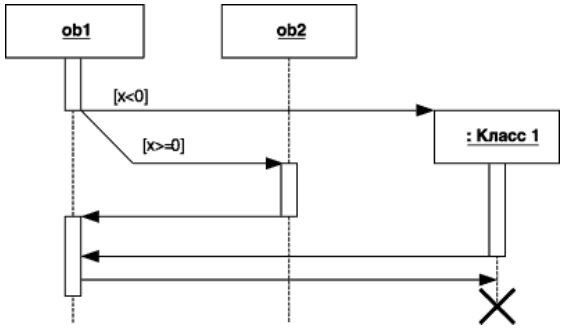


Рис. 6.5. Графическое изображение бинарного ветвления потока управления на диаграмме последовательности

С помощью ветвления можно изобразить и более сложную логику взаимодействия объектов между собой (объект «ob1» на рис.40). Если условий более двух, то для каждого из них необходимо предусмотреть ситуацию единственного выполнения. Описанный ниже пример относится к моделированию взаимодействия программной системы обслуживания клиентов в банке.

Пример: объект «ob1» вызывает выполнение действий у одного из трех других объектов. Условием ветвления может служить сумма снимаемых клиентом средств со своего текущего счета. Если эта сумма превышает 1500$, то могут потребоваться дополнительные действия, связанные с созданием и последующим разрушением объекта «Класса 1». Если же сумма превышает 100$, но не превышает 1500$, то вызывается операция или процедура объекта «ob3». И, наконец, если сумма не превышает 100$, то вызывается операция или процедура объекта «ob2». При этом объекты «ob1», «ob2» и «ob3» постоянно существуют в системе. Последний объект создается от «Класса 1» только в том случае, если справедливо первое из альтернативных условий. В противном случае он может быть никогда не создан.

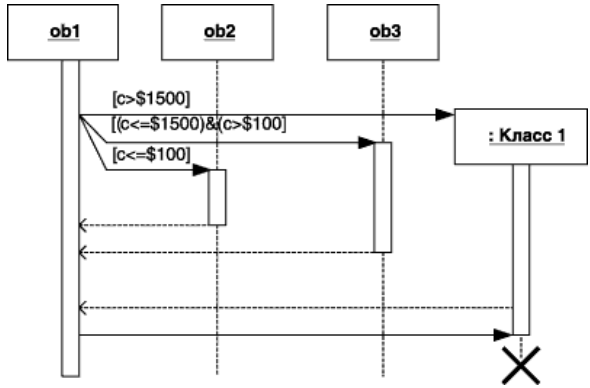


Рис. 40. Графическое изображение тернарного ветвления потока управления на диаграмме последовательности

Объект «ob1» имеет постоянный фокус управления, а все остальные объекты - получают фокус управления только для выполнения ими соответствующих операций. На диаграммах последовательности при записи сообщений также могут использоваться стереотипы. Ниже представлена диаграмма последовательности для описанного выше случая ветвления, дополненная стереотипными значениями отдельных сообщений (рис. 41).

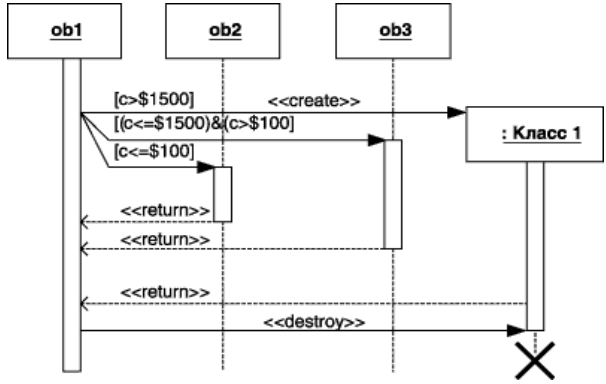


Рис. 41. Диаграмма последовательности со стереотипными значениями сообщений

Сообщения могут иметь собственное имя, в качестве которого выступает имя операции, вызов которой инициируют эти сообщения у принимающего объекта. Тогда со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все равно изображаются. Построение диаграммы последовательности начинают с выделения из всей совокупности классов только тех, объекты которых участвуют во взаимодействии. После все объекты наносятся на диаграмму, с соблюдением порядка инициализации сообщений.

Когда объекты визуализированы, можно приступать к спецификации сообщений, учитывая те операции, которые имеют классы соответствующих объектов, иногда используя их стереотипы.

Для уничтожения объектов, которые создаются на время выполнения своих действий, нужно предусмотреть явное сообщение. Наиболее простые случаи ветвления процесса взаимодействия можно изобразить на одной диаграмме, а в более сложных случаях для моделирования каждой ветви управления может потребоваться отдельная диаграмма последовательности. Общим правилом является визуализация особенностей реализации каждого варианта использования на отдельной диаграмме последовательности.

Пример: Программное средство представляет среду для формирования отчетов по лекционному материалу. Пользователь может вводить свою информацию в отчеты, изменять параметры. После оформления отчетов пользователю предоставлена возможность сохранение отчета. Вывода его на печать и его архивацию.

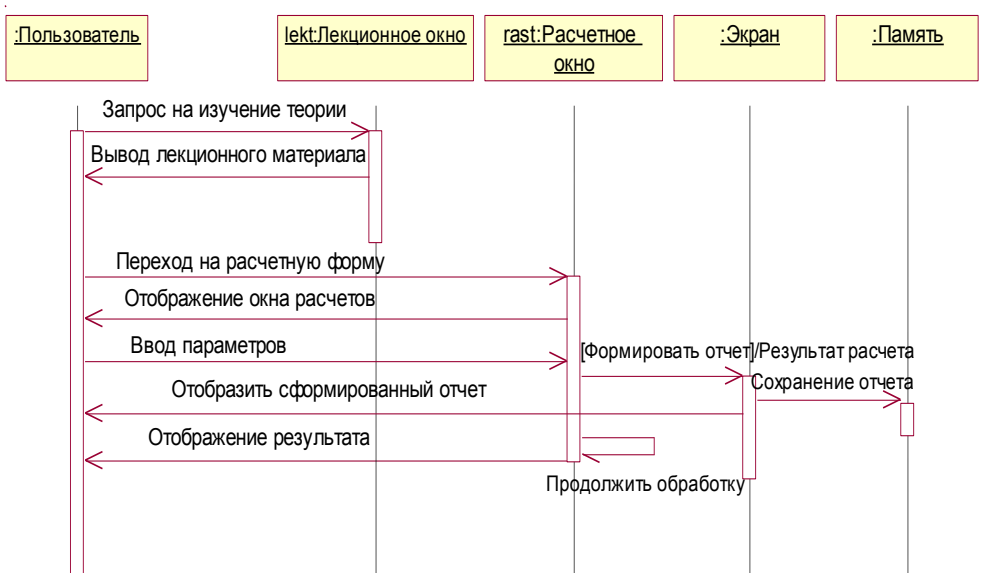


Рис. 42. Пример диаграммы последовательности

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Диаграмма последовательности”.
2. Разработать диаграмму последовательности для произвольной системы индивидуального задания (Приложение А).
3. Оформить отчет, включив в него описание всех компонентов диаграммы последовательности согласно индивидуальному варианту задания, ответы на контрольные вопросы.

Контрольные вопросы

1. Дайте определение понятию «диаграмма последовательности».
2. Опишите назначение диаграммы последовательности.
3. В чем различие анонимного объекта и объекта-сироты?
4. Охарактеризуйте линию жизни объекта. Ее графическое изображение.
5. Для чего предназначен фокус управления? Его графическое изображение.
6. Дайте определение понятию «сообщение». Какие виды сообщений используются в диаграмме последовательности? Когда используется рефлексивное сообщение?
7. Расскажите про ветвления на диаграмме последовательности. Приведите пример ветвления.

ЛАБОРАТОРНАЯ РАБОТА № 5
ДИАГРАММА СОСТОЯНИЙ

Цель работы: закрепление теоретических сведений о диаграмме состояний; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме состояний.

Краткие теоретические сведения

Для представления динамических особенностей взаимодействия элементов модели, в контексте реализации вариантов использования, предназначены диаграммы кооперации и последовательности. Однако для моделирования процессов функционирования большинства сложных систем, особенно систем реального времени, этих представлений недостаточно.

Диаграмма состояний (statechart diagram) - диаграмма, которая представляет конечный автомат, в которой вершины обозначают состояния, а дуги показывают переходы между двумя состояниями.

Назначение диаграммы состояний - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение моделируемой системы в течение всего ее жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей, на основе спецификации их реакции на восприятие некоторых конкретных событий. Системы, которые реагируют на внешние действия от других систем или от пользователей, иногда называют *реактивными*. Если такие действия инициируются в произвольные случайные моменты времени, то говорят об асинхронном поведении модели.

Диаграммы состояний используются для описания поведения отдельных систем и подсистем. Они также могут быть применены для спецификации функциональности экземпляров отдельных классов, т. е. для моделирования всех возможных изменений состояний конкретных объектов. Диаграмма состояний по существу является графом специального вида, который служит для представления конечного автомата.

Диаграммы состояний могут быть вложены друг в друга, образуя вложенные

диаграммы для более детального представления состояний отдельных элементов модели. Для понимания семантики конкретной диаграммы состояний необходимо представлять особенности поведения моделируемой сущности, а также иметь общие сведения из теории конечных автоматов.

Конечный автомат (state machine) - модель для спецификации поведения объекта в форме последовательности его состояний, которые описывают реакцию объекта на внешние события, выполнение объектом действий, а также изменение его отдельных свойств.

В контексте языка UML понятие конечного автомата обладает дополнительной семантикой. Вершинами графа конечного автомата являются состояния и другие типы элементов модели, которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Конечный автомат описывает поведение отдельного объекта в форме последовательности состояний, охватывающих все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет собой конечный автомат.

Основными понятиями, характеризующими конечный автомат, являются состояние и переход. Различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Другими словами, переход объекта из состояния в состояние происходит мгновенно.

В общем случае конечный автомат представляет динамические аспекты моделируемой системы в виде ориентированного графа, вершины которого соответствуют состояниям, а дуги - переходам. При этом поведение моделируется, как последовательное перемещение по графу состояний от вершины к вершине по связывающим их дугам с учетом их ориентации. Для графа состояний системы можно ввести в рассмотрение специальные свойства.

Среди таких свойств - выделение из всей совокупности состояний двух специальных: начального и конечного. Ни в графе состояний, ни на диаграмме состояний время нахождения системы в том или ином состоянии явно не учитывается, однако предполагается, что последовательность изменения состояний упорядочена во времени. Другими словами, каждое последующее состояние может наступить позже предшествующего ему состояния.

Состояние (state) - условие или ситуация в ходе жизненного цикла объекта, в течение которого он удовлетворяет логическому условию, выполняет определенную деятельность или ожидает события.

Состояние может быть задано в виде набора конкретных значений атрибутов объекта некоторого класса, при этом изменение отдельных значений этих атрибутов будет отражать изменение состояния моделируемого объекта или системы в целом. Однако не каждый атрибут класса может характеризовать состояние его объектов. Как правило, имеют значение только те свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения. В этом случае состояние будет характеризоваться некоторым инвариантным условием, включающим в себя только принципиальные для поведения объекта или системы атрибуты классов и их значения.

Такое условие может соответствовать ситуации, когда моделируемый объект находится в состоянии ожидания возникновения внешнего события. В то же время нахождение объекта в некотором состоянии может быть связано с выполнением определенных действий. В последнем случае соответствующая деятельность начинается в момент перехода моделируемого элемента в рассматриваемое состояние, а после и элемент может покинуть данное состояние в момент завершения этой деятельности.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рис. 43). Этот прямоугольник может быть разделен на две секции. Если указана лишь одна секция, то в ней записывается только имя состояния (рис. 43, а). В противном случае в первой из них записывается имя состояния, а во второй - список некоторых внутренних действий или переходов в данном состоянии (рис. 43, б). При этом под действием в языке UML понимают некоторую атомарную операцию, выполнение которой приводит к изменению состояния или возврату некоторого значения (например, "истина" или "ложь").

Рис. 43. Графическое изображение состояний на диаграмме состояний

Имя у состояния может отсутствовать, т. е. оно необязательно для некоторых состояний. В этом случае состояние является *анонимным*. Если на одной диаграмме состояний несколько анонимных состояний, то все они должны различаться между собой.

Действие (action) - спецификация выполнимого утверждения, которая образует абстракцию вычислительной процедуры.

Действие обычно приводит к изменению состояния системы, и может быть реализовано посредством передачи сообщения объекту, модификацией связи или значения атрибута. Для ряда состояний может потребоваться дополнительно указать действия, которые должны быть выполнены моделируемым элементом. Для этой цели служит добавочная секция в обозначении состояния, содержащая перечень внутренних действий или деятельность, которые производятся в процессе нахождения моделируемого элемента в данном состоянии. Каждое действие записывается в виде отдельной строки и имеет следующий формат:

```
<метка действия '/ ' выражение действия>
```

Метка действия указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия. При этом выражение действия может использовать любые атрибуты и связи, принадлежащие области имен или контексту моделируемого объекта. Если список выражений действия пустой, то метка действия с разделителем в виде наклонной черты '/' не указывается.

Перечень меток действий в языке UML фиксирован, причем эти метки не могут быть использованы в качестве имен событий:

Входное действие (entry action) - действие, которое выполняется в момент перехода в данное состояние. Обозначается с помощью ключевого слова - метки действия entry, которое указывает на то, что следующее за ней выражение действия должно быть выполнено в момент входа в данное состояние.

Действие выхода (exit action) - действие, производимое при выходе из данного состояния. Обозначается с помощью ключевого слова - метки действия exit, которое указывает на то, что следующее за ней выражение действия должно быть выполнено в момент выхода из данного состояния.

Внутренняя деятельность (do activity) - выполнение объектом операций или процедур, которые требуют определенного времени. Обозначается с помощью ключевого слова - метки деятельности do, которое специфицирует так называемую "дудеятельность", выполняемую в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не будет прервано внешним событием. При нормальном завершении внутренней деятельности генерируется соответствующее событие. Во всех остальных случаях метка действия идентифицирует событие, которое запускает соответствующее выражение действия. Эти события называются внутренними переходами. Семантически они эквивалентны переходам в само это состояние, за исключением той особенности, что выход из этого состояния или повторный вход в него не происходит. Это означает, что действия входа и выхода не производятся. При этом выполнение внутренних действий в состоянии не может быть прервано никакими внешними событиями, в отличие от внутренней деятельности, выполнение которой требует определенного времени.

Пример: аутентификацию клиента для доступа к ресурсам моделируемой информационной системы (рис. 44). Список внутренних действий в данном состоянии может включать следующие действия. Первое действие - входное, которое выполняется при входе в это состояние и связано с получением строки символов, соответствующих паролю клиента. Далее выполняется деятельность по проверке введенного клиентом пароля. При успешном завершении этой проверки выполняется действие на выходе, которое отображает меню доступных для клиента опций.

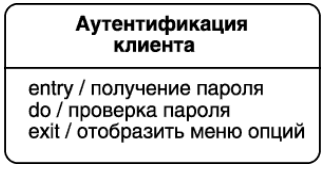


Рис. 44. Пример состояния с непустой секцией внутренних действий

Кроме обычных состояний на диаграмме состояний могут размещаться псевдосостояния.

Псевдосостояние (pseudo-state) - вершина в конечном автомате, которая имеет форму состояния, но не обладает поведением.

Примерами псевдосостояний, которые определены в языке UML, являются начальное и конечное состояния.

Начальное состояние (start state) - разновидность псевдосостояния, обозначающее начало выполнения процесса изменения состояний конечного автомата или нахождения моделируемого объекта в составном состоянии.

В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (рис. 45, а), из которого может только выходить стрелка-переход.

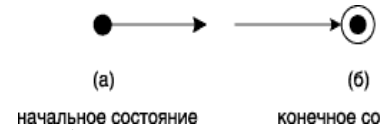


Рис. 45. Графическое изображение начального и конечного состояний

На самом верхнем уровне представления объекта переход из начального состояния может быть помечен событием создания (инициализации) данного объекта. В противном случае этот переход никак не помечается. Если этот переход не помечен, то он является первым переходом на диаграмме состояний в следующее за ним состояние. Каждая диаграмма или под-диаграмма состояний должна иметь единственное начальное состояние.

Конечное состояние (final state) - разновидность псевдосостояния, обозначающее прекращение процесса изменения состояний конечного автомата или нахождения моделируемого объекта в составном состоянии.

В этом состоянии должен находиться моделируемый объект или система по умолчанию после завершения работы конечного автомата. Оно служит для указания на диаграмме состояний графической области, в которой завершается процесс изменения состояний или жизненный цикл данного объекта.

Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 45, б), в которую может только входить стрелка-переход. Каждая диаграмма состояний или подсостояний может иметь несколько конечных состояний, при этом все они считаются эквивалентными на одном уровне вложенности состояний.

Пребывание объекта в первом состоянии может сопровождаться выполнением некоторых внутренних действий или деятельности. При этом изменение текущего состояния объекта будет возможно либо после завершения этих действий, либо при возникновении некоторых внешних событий. В обоих случаях говорят, что происходит переход объекта из одного состояния в другое.

Переход (transition) - отношение между двумя состояниями, которое указывает на то, что объект в первом состоянии должен выполнить определенные действия и перейти во второе состояние.

Переход осуществляется при наступлении некоторого события: окончания выполнения деятельности (do activity), получении объектом сообщения или приемом сигнала. На переходе указывается имя события, а также действия, производимые объектом в ответ на внешние события при переходе из одного состояния в другое.

Переход может быть направлен в то же состояние, из которого он выходит. В этом случае его называют переходом в себя. Исходное и целевое состояния перехода в себя совпадают. Этот переход изображается петлей со стрелкой и отличается от внутреннего перехода. При переходе в себя объект покидает исходное состояние, а затем снова входит в него. При этом всякий раз выполняются внутренние действия, специфицированные метками entry и exit.

Срабатывание <перехода> (fire) - выполнение перехода из одного состояния в другое.

Срабатывание перехода может зависеть не только от наступления события, но и от выполнения определенного условия, называемого сторожевым. Объект перейдет из одного состояния в другое в том случае, если произошло указанное событие и сторожевое условие приняло значение "истина".

До срабатывания перехода объект находится в предыдущем от него состоянии, называемым исходным, или в источнике (не путать с начальным состоянием - это разные понятия), а после его срабатывания объект находится в последующем от него состоянии (целевом состоянии).

На диаграмме состояний переход изображается сплошной линией со стрелкой, которая выходит из исходного состояния и направлена в целевое состояние. Каждый переход может быть помечен строкой текста, которая имеет следующий общий формат: `<имя события> '('<список параметров>')' '['<сторожевое условие>']' <выражение действия>`

Событие (event) - спецификация существенных явлений в поведении системы, которые имеют местоположение во времени и пространстве.

Формально, событие представляет собой спецификацию 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 события нельзя уже вернуться к предыдущим, если такая возможность явно не предусмотрена в модели.

Семантика понятия события фиксирует внимание на внешних проявлениях качественных изменений, происходящих при переходе моделируемого объекта из состояния в состояние.

Например, при включении электрического переключателя происходит событие, в результате которого комната освещается. После успешного ремонта компьютера также происходит немаловажное событие - восстановление его работоспособности. Если поднять трубку обычного телефона, то, в случае его исправности, мы ожидаем услышать тоновый сигнал. Это тоже является событием.

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. В зависимости от вида происходящих событий - стимулов в языке UML различают два типа переходов: триггерные и нетриггерные.

Переход называется **триггерным**, если его специфицирует событие-триггер, связанное с внешними условиями по отношению к рассматриваемому состоянию.

В этом случае рядом со стрелкой триггерного перехода обязательно указывается имя события в форме строки текста, начинающейся со строчной буквы. Наиболее

часто в качестве имен триггерных переходов задают имена операций, вызываемых у тех или иных объектов системы. После имени такого события следуют круглые скобки для явного задания параметров соответствующей операции. Если таких параметров нет, то список параметров со скобками может отсутствовать. Например, переход на рис. 46, а, является триггерным, поскольку с ним связано конкретное событие-триггер, происходящее асинхронно при срабатывании некоторого датчика.

Переход называется **нетриггерным**, если он происходит по завершении выполнения ду-деятельности в данном состоянии.

Нетриггерные переходы часто называют переходами по завершении ду-деятельности. Для них рядом со стрелкой перехода не указывается никакого имени события, а в исходном состоянии должна быть описана внутренняя ду-деятельность, по окончании которой произойдет тот или иной нетриггерный переход.

Рис. 46. Графическое изображение триггерного (а) и нетриггерного (б) переходов на диаграмме состояний

Сторожевое условие (guard condition) - логическое условие, записанное в прямых скобках и представляющее собой булевское выражение.При этом булевское выражение должно принимать одно из двух взаимно исключающих значений: "истина" или "ложь". Из контекста диаграммы состояний должна явно следовать семантика этого выражения, а для записи выражения может использоваться обычный язык, псевдокод или язык программирования.

Дополнение триггерных и нетриггерных переходов сторожевыми условиями позволяет явно специфицировать семантику их срабатывания.

Если сторожевое условие принимает значение "истина", то соответствующий переход при наступлении события-триггера или завершении деятельности может сработать, в результате чего объект перейдет в целевое состояние.

Если же сторожевое условие принимает значение "ложь", то переход не может сработать, даже если произошло событие-триггер или завершилась деятельность в исходном состоянии.

Очевидно, в случае невыполнения сторожевого условия моделируемый объект или система останется в исходном состоянии. Однако вычисление истинности сторожевого условия в модели происходит только после возникновения ассоциированного с ним события-триггера или завершения деятельности, которые инициируют соответствующий переход.

Поскольку общее количество выходящих переходов из любого состояния не ограничено, не исключена ситуация, когда из одного состояния могут выходить несколько переходов с идентичным событием-триггером. Каждый такой переход должен содержать собственное сторожевое условие, при этом никакие два или более сторожевых условий не должны одновременно принимать значение "истина". В противном случае на диаграмме состояний возникнет конфликт триггерных переходов, что делает несостоятельной (ill formed) модель системы в целом.

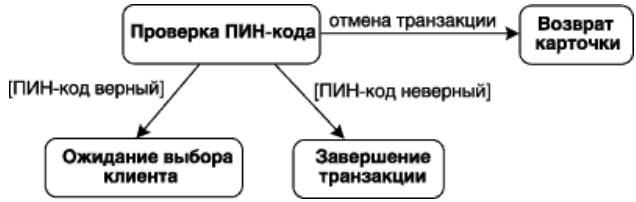


Рис. 47. Триггерные и нетриггерные переходы на диаграмме состояний

Изображенный фрагмент диаграммы состояний (рис. 47) моделирует изменение состояний банкомата при проверке ПИН-кода. Нетриггерные переходы на данной диаграмме помечены сторожевыми условиями, которые исключают конфликт между ними. Что касается триггерного перехода, помеченного событием отмена транзакции, то он происходит независимо от проверки ПИН-кода в том случае, когда клиент решил отказаться от ввода ПИН-кода.

Выражение действия (action expression) представляет собой вызов операции или передачу сообщения, имеет атомарный характер и выполняется сразу после срабатывания соответствующего перехода до начала действий в целевом состоянии. Выражение действия выполняется в том и только в том случае, когда переход срабатывает.

Атомарность действия означает, что оно не может быть прервано никаким другим действием до тех пор, пока не закончится его выполнение. Данное действие может оказывать влияние как на сам объект, так и на его окружение, если это с очевидностью следует из контекста модели. Данное выражение записывается после знака "/" в строке текста, присоединенной к соответствующему переходу.

В общем случае, выражение действия может содержать целый список отдельных действий, разделенных символом ";". Обязательное требование - все действия из списка должны четко различаться между собой и следовать в порядке их записи. На синтаксис записи выражений действия не накладывается никаких ограничений. Главное - их запись должна быть понятна разработчикам модели и программистам. Поэтому чаще всего выражения записывают на одном из языков программирования, который предполагается использовать для реализации модели.

В качестве примера выражения действия перехода (рис. 48) может служить отображение сообщения на экране банкомата в том случае, когда запрашиваемая клиентом сумма превосходит остаток на его счету. В случае если кредит не превышен, то происходит переход в состояние получения наличных.

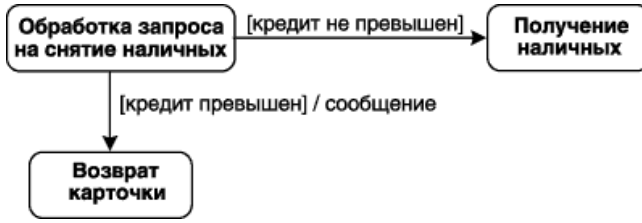


Рис. 48. Выражение действия перехода на диаграмме состояний

Формализм конечных автоматов допускает вложение одних конечных автоматов в другие для уточнения внутренней структуры отдельных более общих состояний. В этом случае вложенные конечные автоматы получили название конечных подавтоматов. Подавтоматы могут использоваться для внутренней спецификации процедур и функций, реализация которых обусловливает поведение моделируемой системы или объекта.

Моделирование сложных объектов и систем связано с многоуровневым представлением их состояний. В этом случае возникает необходимость детализировать отдельные состояния, сделав их составными.

Составное состояние (composite state) - сложное состояние, которое состоит из других вложенных в него состояний (состояние-композит). Вложенные состояния выступают по отношению к составному состоянию как **подсостояния** (substate). И хотя между ними имеет место отношение композиции, графически все вершины диаграммы, которые соответствуют вложенным состояниям, изображаются внутри символа составного состояния (рис. 49). В этом случае размеры графического символа составного состояния увеличиваются, так чтобы вместить в себя все подсостояния.

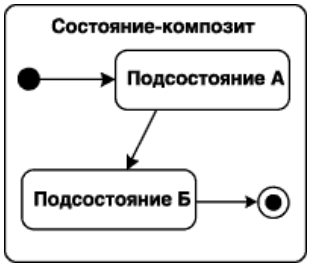


Рис. 49. Графическое представление составного состояния с двумя вложенными в него последовательными подсостояниями

Составное состояние может содержать или несколько последовательных подсостояний, или несколько параллельных конечных подавтоматов. Каждое состояние-композит может уточняться только одним из указанных способов. При этом любое из подсостояний, в свою очередь, может быть состоянием-композитом и содержать внутри себя другие вложенные подсостояния. Количество уровней вложенности составных состояний в языке UML не фиксировано.

Последовательные_подсостояния (sequential substates) - вложенные состояния

состояния-композита, в рамках которого в каждый момент времени объект может находиться в одном и только одном подсостоянии.

Поведение объекта в этом случае представляет собой последовательную смену подсостояний, от начального до конечного. Моделируемый объект или система продолжает находиться в составном состоянии, тем не менее, введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты его внутреннего поведения.

В качестве примера моделируемой системы стоит рассмотреть обычный телефонный аппарат. Он может находиться в различных состояниях, в частности в состоянии дозвона до абонента. Очевидно, для того чтобы позвонить, необходимо снять телефонную трубку, услышать тоновый сигнал, после чего набрать нужный телефонный номер. Таким образом, состояние дозвона до абонента является составным и состоит из двух последовательных подсостояний: Телефонная трубка поднята и Набор телефонного номера. Фрагмент диаграммы состояний для этого примера содержит одно состояние-композит, которое состоит из двух последовательных подсостояний (рис. 50).

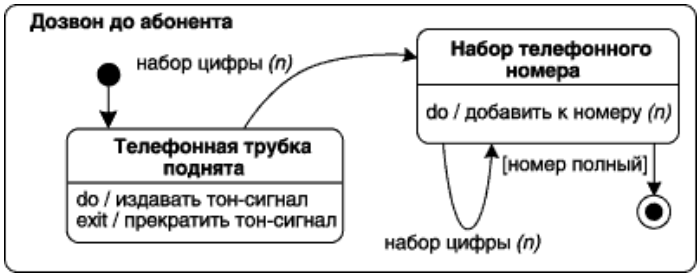


Рис. 50. Пример составного состояния с двумя вложенными последовательными подсостояниями

Некоторых пояснений могут потребовать переходы. Два из них специфицируют событие-триггер, которое имеет имя: набор цифры(n) с параметром n. В качестве параметра, как нетрудно предположить, выступает отдельная цифра на диске телефонного аппарата. Переход из начального подсостояния не содержит никакой строки текста. Последний переход в конечное подсостояние также не имеет события-триггера, но имеет сторожевое условие, проверяющее полноту набранного номера абонента. Только в случае истинности этого условия телефонный аппарат может перейти в конечное состояние для состояния-композита Дозвон до абонента.

Каждое составное состояние должно содержать в качестве вложенных состояний начальное и конечное состояния. При этом начальное подсостояние является исходным, когда происходит переход объекта в данное составное состояние. Если составное состояние содержит внутри себя конечное состояние, то переход в это вложенное конечное состояние означает завершение нахождения объекта в данном составном состоянии. Важно помнить, что для последовательных подсостояний начальное и конечное состояния должны быть единственными в каждом составном состоянии.

Это можно объяснить следующим образом. Каждая совокупность вложенных последовательных подсостояний представляет собой конечный подавтомат того конечного автомата, которому принадлежит рассматриваемое составное состояние. Поскольку каждый конечный автомат может иметь по определению единственное начальное и единственное конечное состояния, то для любого его конечного подавтомата это условие также должно выполняться.

Параллельные подсостояния (concurrent substates) - вложенные состояния, используемые для спецификации двух и более конечных подавтоматов, которые могут выполняться параллельно внутри составного состояния.

Каждый из конечных подавтоматов занимает некоторую графическую область внутри составного состояния, которая отделяется от остальных горизонтальной пунктирной линией. Если на диаграмме состояний имеется составное состояние с вложенными параллельными подсостояниями, то объект может одновременно находиться в каждом из этих подсостояний.

Отдельные параллельные подсостояния могут, в свою очередь, состоять из нескольких последовательных подсостояний (рис. 51). В этом случае по определению моделируемый объект может находиться только в одном из последовательных подсостояний каждого подавтомата. Таким образом, для фрагмента диаграммы состояний (рис. 51) допустимо одновременное нахождение объекта только в следующих подсостояниях: (А, В, Г), (Б, В, Г), (А, В, Д), (Б, В, Д).

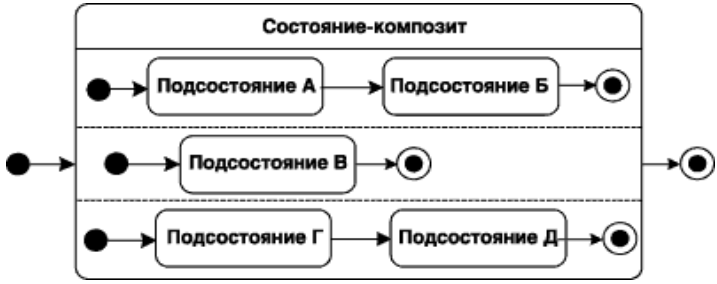


Рис. 51. Графическое изображение состояния-композита с вложенными параллельными подсостояниями

Несовместимое подсостояние (disjoint substate) - подсостояние, в котором подсистема не может находиться одновременно с другими подсостояниями одного и того же составного состояния.

В этом контексте недопустимо нахождение объекта одновременно в несовместимых подсостояниях (А, Б, В) или (В, Г, Д).

Поскольку каждый регион вложенного состояния специфицирует некоторый конечный подавтомат, то для каждого из вложенных конечных подавтоматов могут быть определены собственные начальное и конечное состояния.

При переходе в данное составное состояние каждый из конечных подавтоматов оказывается в своем начальном состоянии. Далее происходит параллельное выполнение каждого из этих конечных подавтоматов, причем выход из составного состояния будет возможен лишь в том случае, когда все конечные подавтоматы будут на-

ходиться в своих конечных состояниях. Если какой-либо из конечных подавтоматов пришел в свое финальное состояние раньше других, то он должен ожидать, пока и другие подавтоматы не придут в свои финальные состояния.

В некоторых случаях бывает желательно скрыть внутреннюю структуру составного состояния. Например, отдельный конечный подавтомат, специфицирующий составное состояние, может быть настолько большим по масштабу, что его визуализация затруднит общее представление диаграммы состояний. В подобной ситуации допускается не раскрывать на исходной диаграмме состояний данное составное состояние, а указать в правом нижнем углу специальный символ-пиктограмму (рис. 52).

В последующем диаграмма состояний для соответствующего конечного подавтомата может быть изображена отдельно от основной диаграммы с необходимыми комментариями.

Рис. 52. Составное состояние со скрытой внутренней структурой и специальной пиктограммой

Обычный конечный автомат не позволяет учитывать предысторию в процессе моделирования поведения систем и объектов. Однако функционирование ряда систем основано на возможности выхода из отдельного состояния-композита с последующим возвращением в это же состояние.

Может оказаться необходимым учесть ту часть деятельности, которая была выполнена на момент выхода из этого состояния-композита, чтобы не начинать ее выполнение сначала. Для этой цели в языке UML существует историческое состояние.

Историческое состояние (history state) - псевдосостояние, используемое для запоминания того из последовательных подсостояний, которое было текущим в момент выхода из составного состояния.

Историческое состояние применяется только в контексте составного состояния. При этом существует две разновидности исторического состояния: неглубокое или недавнее и глубокое или давнее (рис. 53).

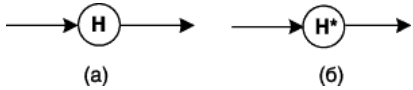


Рис. 53. Графическое изображение недавнего (а) и давнего (б) исторического состояния

Неглубокое историческое состояние (shallow history state) обозначается в форме небольшой окружности, в которую помещена латинская буква "H" (рис. 53, а). Это

состояние обладает следующей семантикой. Во-первых, оно является первым подсостоянием в составном состоянии, и переход извне в рассматриваемое составное состояние должен вести непосредственно в данное историческое состояние. Во-вторых, при первом попадании в неглубокое историческое состояние оно не хранит никакой истории. Другими словами, при первом переходе в недавнее историческое состояние оно заменяет собой начальное состояние соответствующего конечного подавтомата.

Далее могут последовательно изменяться вложенные подсостояния. Если в некоторый момент происходит выход из составного состояния (например, в случае наступления некоторого события), то рассматриваемое историческое состояние запоминает то из подсостояний, которое было текущим на момент выхода из данного составного состояния. При последующем входе в это составное состояние неглубокое историческое подсостояние имеет непустую историю и сразу отправляет конечный подавтомат в запомненное подсостояние, минуя все предшествующие ему подсостояния.

Историческое состояние теряет свою историю в тот момент, когда конечный подавтомат доходит до своего конечного состояния. При этом неглубокое историческое состояние запоминает историю только того конечного подавтомата, к которому оно относится. Другими словами, этот тип псевдосостояния способен запомнить историю только одного с ним уровня вложенности.

Глубокое историческое состояние (deep history state или состояние глубокой истории) также обозначается в форме небольшой окружности, в которую помещена латинская буква "H" с дополнительным символом "*" (рис. 53, б), и служит для запоминания всех подсостояний любого уровня вложенности для исходного составного состояния.

В отдельных случаях возникает необходимость явно показать ситуацию, когда переход может иметь несколько исходных состояний или целевых состояний. Такой переход получил название - **параллельный переход**. Введение в рассмотрение параллельных переходов может быть обусловлено необходимостью синхронизировать и/или разделить отдельные процессы управления на параллельные нити без спецификации дополнительной синхронизации в параллельных конечных подавтоматах.

Графически такой переход изображается вертикальной черточкой, аналогично обозначению перехода в известном формализме сетей Петри. Если параллельный переход имеет две или более исходящих из него дуг (рис. 54, а), то его называют разделением (fork). Если же он имеет две или более входящие дуги (рис. 54, б), то его называют слиянием (join). Текстовая строка спецификации параллельного перехода записывается рядом с черточкой и относится ко всем входящим или исходящим дугам.

Срабатывание параллельного перехода: в первом случае происходит разделение составного конечного автомата на два конечных подавтомата, образующих параллельные ветви вложенных подпроцессов. При этом после срабатывания перехода-разделения моделируемая система или объект одновременно будет находиться во всех целевых подсостояниях этого параллельного перехода (подсостояния 1 и 2). Далее процесс изменения состояний будет протекать согласно ранее рассмотренным правилам для составных состояний.

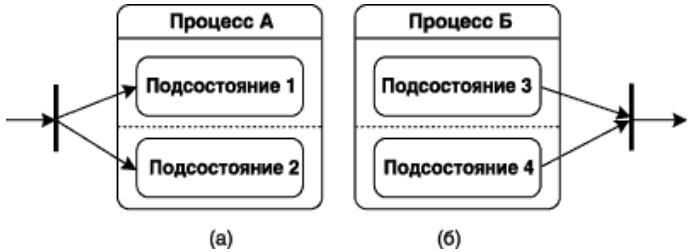


Рис. 54. Графическое изображение перехода-разделения в параллельные подсостояния (а) и перехода-слияния из параллельных подсостояний (б)

Во втором случае переход-слияние срабатывает, если имеет место событие-триггер для всех исходных состояний этого перехода, и выполнено (при его наличии) сторожевое условие. При срабатывании перехода-слияния одновременно покидаются все исходные подсостояния перехода (подсостояния 3 и 4) и происходит переход в целевое состояние. При этом каждое из исходных подсостояний перехода должно принадлежать отдельному конечному подавтомату, входящему в составной конечный автомат (процессу Б).

Переход, стрелка которого соединена с границей составного состояния, обозначает переход в это составное состояние (переход a на рис. 55). Он эквивалентен переходу в начальное состояние каждого из конечных подавтоматов (единственному на рис. 55), входящих в состав данного состояния-композита.

Переход f, выходящий из составного состояния (рис. 55), относится к каждому из вложенных состояний. Это означает, что моделируемая система или объект при наступления события f может покинуть данное составное состояние, находясь в любом из его вложенных состояний В и Г.

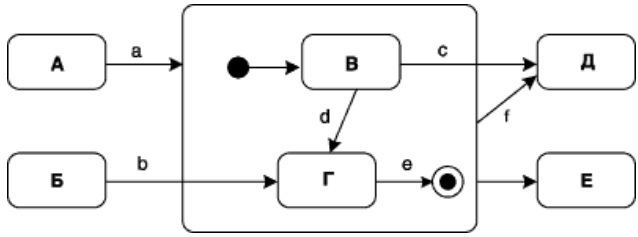


Рис. 55. Варианты переходов в составное состояние и из составного состояния

Иногда желательно реализовать ситуацию, когда выход из отдельного вложенного состояния соответствовал бы также выходу из составного состояния. В этом случае изображают переход, который непосредственно выходит из вложенного состояния и пересекает границу состояния-композита (переход c на рис. 55). Аналогично, допускается изображение переходов, входящих извне состояния-композита в отдельное вложенное состояние (переход b на рис. 55).

Переход d является внутренним для рассматриваемого состояния-композита и никак не влияет на выход из состояния-композита. Выход из данного составного

состояния также возможен при наступлении события e, которое приводит в его конечное состояние, а из него - в состояние E, находящееся вне данного состояния-композита.

Состояние синхронизации (synch state) - псевдосостояние в конечном автомате, которое используется для синхронизации параллельных областей конечного автомата.

Синхронизирующее состояние обозначается небольшой окружностью, внутри которой помещен символ звездочки "*". Оно используется совместно с переходом-слиянием или переходом-разделением для того, чтобы явно указать события в других конечных подавтоматах, оказывающие непосредственное влияние на поведение данного подавтомата.

Так, например, при включении компьютера с некоторой сетевой операционной системой параллельно начинается выполнение нескольких процессов. В частности, происходит проверка пароля пользователя и запуск различных служб. При этом работа пользователя на компьютере станет возможной только в случае успешной его аутентификации, в противном случае компьютер может быть выключен.

Рассмотренные особенности синхронизации этих параллельных процессов учтены на соответствующей диаграмме состояний с помощью синхронизирующего состояния (рис. 56).

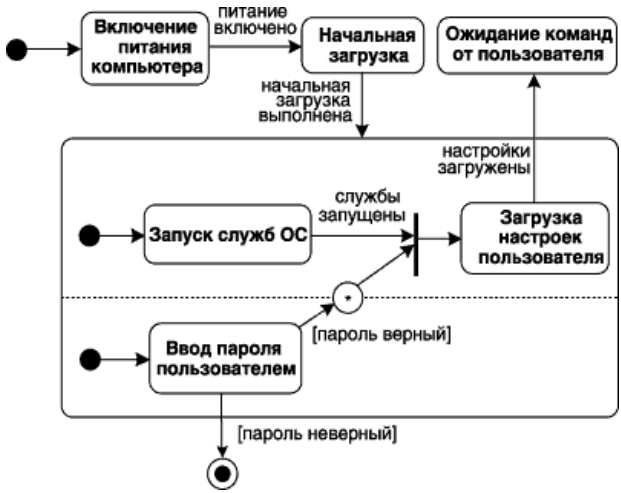


Рис. 56. Диаграмма состояний для примера включения компьютера

Наличие у системы нескольких состояний, отличающихся от простых, таких как "исправен - неисправен", "активен - неактивен", "ожидание - реакция на внешние действия", служит признаком необходимости построения диаграммы состояний. В качестве начального варианта диаграммы состояний, если нет очевидных соображений по поводу состояний объекта, можно воспользоваться подобными состояниями, в качестве составных, уточняя их (детализируя их внутреннюю структуру) по мере рассмотрения логики поведения моделируемой системы или объекта.

При выделении состояний и переходов следует помнить, что длительность срабатывания отдельных переходов должна быть существенно меньшей, чем нахождение моделируемых элементов в соответствующих состояниях. Каждое из состояний должно характеризоваться определенной устойчивостью во времени. Другими словами, из каждого состояния на диаграмме не может быть самопроизвольного перехода в какое бы то ни было другое состояние.

Все переходы должны быть явно специфицированы, в противном случае построенная диаграмма состояний является либо неполной (неадекватной), либо ошибочной с точки зрения нотации языка UML (ill formed).

При разработке диаграммы состояний нужно постоянно следить, чтобы объект в каждый момент мог находиться только в единственном состоянии. Если это не так, то данное обстоятельство может быть как следствием ошибки, так и неявным признаком наличия параллельности поведения у моделируемого объекта. В последнем случае следует явно специфицировать необходимое число конечных подавтоматов, вложив их в то составное состояние, которое характеризуется нарушением условия одновременности.

Пример: Программное средство представляет собой базу данных и обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствие с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

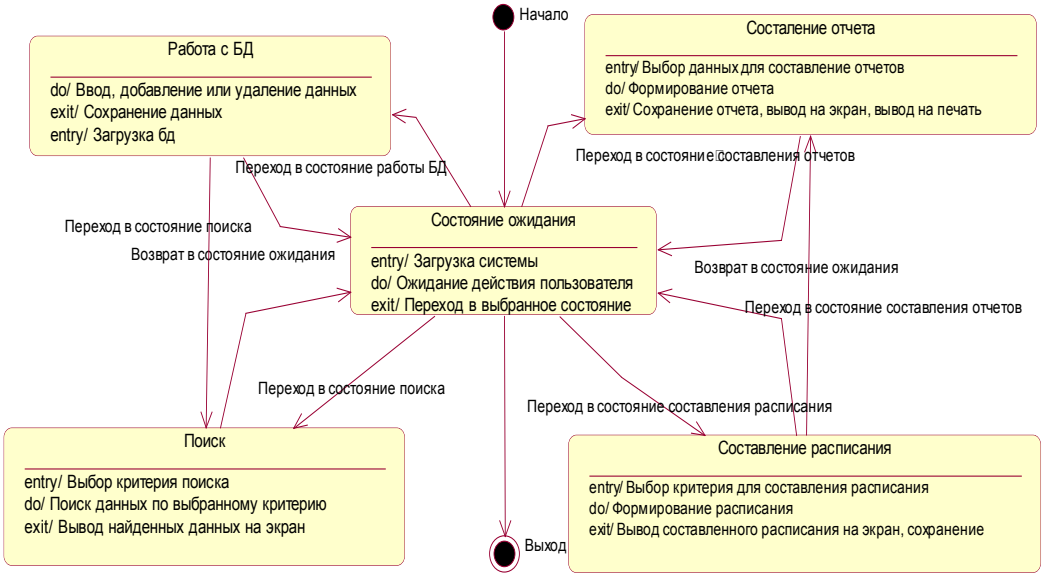


Рис. 57. Пример диаграммы состояний

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Диаграмма состояний”.
2. Разработать диаграмму состояний для произвольной системы индивидуального задания (Приложение А).
3. Оформить отчет, включив в него описание всех компонентов диаграммы состояний согласно индивидуальному варианту задания, ответы на контрольные вопросы.

Контрольные вопросы

1. Дайте определение понятию «диаграмма состояний».
2. Опишите назначение диаграммы состояний.
3. Дайте определение понятию «конечный автомат».
4. Дайте определение понятиям «состояния», «действие», «псевдосостояние». Графическое изображение состояния.
5. Что представляет собой переход? Какие бывают переходы, в чем их различие?
6. Дайте определение понятию «событие».
7. Дайте определения следующим понятиям: «составное состояние», «последовательные подсостояния», «параллельные подсостояния», «несовместимое подсостояние», «историческое состояние», «параллельный переход», «состояние синхронизации».

ЛАБОРАТОРНАЯ РАБОТА № 6
ДИАГРАММА ДЕЯТЕЛЬНОСТИ

Цель работы: закрепление теоретических сведений о диаграмме деятельности; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме деятельности.

Краткие теоретические сведения

При моделировании поведения проектируемой или анализируемой программной системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и процедурной реализации выполняемых системой операций. Для этой цели, как правило, используются блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных процедур или элементарных операций, которые в совокупности приводят к получению желаемого результата. С увеличением сложности системы строгое соблюдение определенной последовательности выполняемых действий приобретает большое значение.

Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности

также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления деятельности и действий, а также в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некой операции, а переход в следующее состояние происходит только после завершения выполнения этой операции. Диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия или деятельности, а дугами - переходы от одного состояния действия к другому.

Диаграммы деятельности (activity diagram)- частный случай диаграмм состояний. Они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних действий и деятельности. 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 определенного класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML *деятельность* представляет собой совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к результату или действию. 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. Диаграмма деятельности предназначена для моделирования поведения систем, хотя время в явном виде отсутствует на этой диаграмме. Ситуация здесь во многом аналогична диаграмме состояний, однако имеет ряд особенностей.

Состояние деятельности (activity state) - состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определенного времени. Переход из состояния деятельности происходит после выполнения специфицированной в нем ду-деятельности, при этом ключевое слово do в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным.

Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными, то есть могут быть прерваны. Предполагается, что для их завершения требуется заметное время.

Состояние деятельности можно представлять себе как составное состояние, поток управления которого включает только другие состояния деятельности и действий.

Рис. 58. Графическое изображение состояний деятельности действия

Состояние действия (action state) - специальный случай состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Переход из состояния действия происходит после завершения входного действия. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Использование состояния действия заключается в моделировании шага выполнения алгоритма. Графически состояния деятельности и действия изображаются одинаковой фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами. Внутри этой фигуры записывается имя состояния деятельности или действия в форме выражения, которое должно быть уникальным в пределах одной диаграммы деятельности.

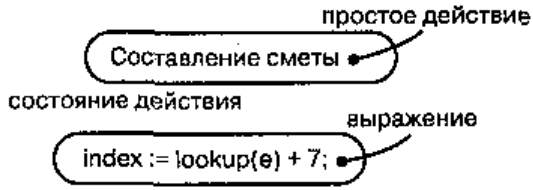


Рис. 59. Графическое изображение состояний действия на диаграмме деятельности

Действие может быть записано на естественном языке, псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами (рис. 59). Если же действие может быть представлено в формальном виде, то оно записывается на том языке программирования, на котором предполагается реализовывать разрабатываемый проект.

Когда возникает необходимость представить на диаграмме деятельности сложное действие, состоящее из нескольких более простых. то используют специальное обозначение так называемого состояния под-деятельности. **Состояние под-деятельности** (subactivity state) - состояние в графе деятельности, которое служит для представления неатомарной последовательности шагов процесса. Это состояние является графом деятельности и обозначается специальной пиктограммой в правом нижнем углу символа состояния действия (рис. 60). Данная конструкция может применяться к любому элементу языка UML, который поддерживает "вложенность" своей структуры. При этом пиктограмма может быть дополнительно помечена типом вложенной структуры.

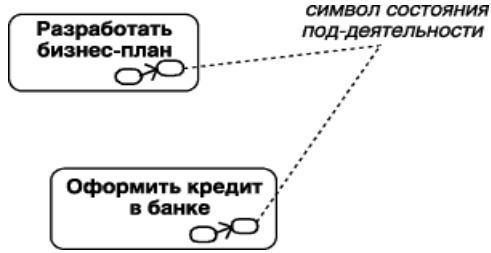


Рис. 60. Графическое изображение состояния под-деятельности

Каждая диаграмма деятельности должна иметь единственное начальное и конечное состояния. Они имеют такие же обозначения, как и на диаграмме состояний. При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии.

Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз или слева направо. В этом случае начальное состояние будет изображаться в верхней или левой части диаграммы, а конечное - в ее нижней или правой части. В интересах удобства визуального представления на диаграмме деятельности допускается изображать несколько конечных состояний. В этом случае все их принято считать эквивалентными друг другу.

Переход на диаграмме деятельности аналогичен переходу на диаграмме состояний. При построении диаграммы деятельности используются только нетриггерные переходы, т.е. такие, которые происходят сразу после завершения деятельности или выполнения соответствующего действия. Такой переход передает управление в последующее состояние сразу, как только закончится действие или деятельность в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то его можно никак не помечать. Если же таких переходов несколько, то при моделировании последовательной деятельности запускается только один из них. В этом случае для каждого из таких переходов должно быть явно записано собственное сторожевое условие в прямых скобках.

При этом для всех выходящих из некоторого состояния деятельности переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения промежуточного результата. Такая ситуация получила название ветвления, а для ее обозначения применяется специальный символ решения.

Графически ветвление на диаграмме деятельности обозначается символом решения (decision), изображаемого в форме небольшого ромба, внутри которого нет никакого текста (рис. 61). В ромб может входить только одна стрелка от того состояния действия, после выполнения, которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа решения. Выходящих стрелок может быть две или более. Для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

Для графического объединения альтернативных ветвей на диаграмме деятельности рекомендуется использовать аналогичный символ в форме ромба, который в этом случае называют соединением (merge). Наличие этого символа, внутри которого также не записывается никакого текста, упрощает визуальный контроль логики выполнения процедурных действий на диаграмме деятельности (рис. 61 внизу). Входящих стрелок у символа соединения может быть несколько, они исходят от состояний действия, принадлежащих к одной из взаимно исключающих ветвей. Выходить из ромба соединения может только одна стрелка, при этом ни входящие, ни выходящая стрелки не должны содержать сторожевых условий. Исключением явля-

ется ситуация, когда с целью сокращения диаграммы объединяют символ решения с символом соединения.

Пример: моделируется ситуация, возникающая в супермаркетах при оплате товаров. Как правило, заплатить за покупки можно либо наличными, либо по кредитной карточке. Если покупателем выбран вариант оплаты по кредитной карточке, то проверяется сумма баланса предъявленной к оплате кредитной карточки. При этом оплата происходит только в том случае, если общая стоимость приобретаемых товаров не превышает суммы баланса этой карточки. В противном случае оплаты не происходит, и товар остается у продавца.

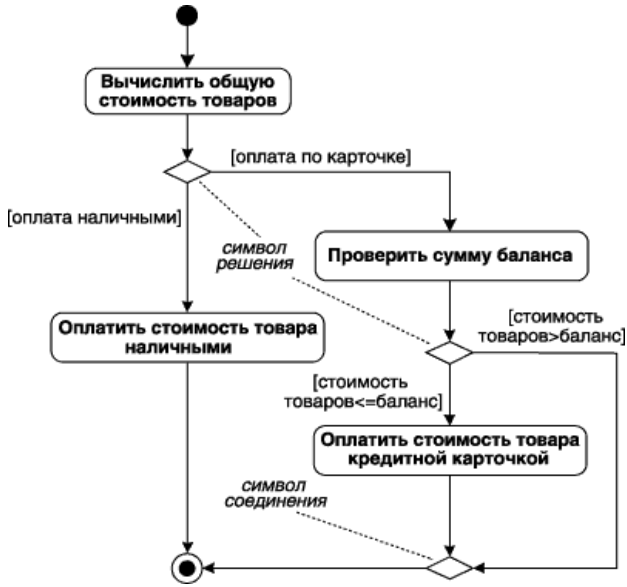


Рис. 61. Различные варианты ветвлений на диаграмме деятельности

На диаграмме деятельности изображаются параллельные процессы, поскольку распараллеливание вычислений существенно повышает быстродействие программных систем. В диаграммах деятельности для изображения параллельных процессов используется специальный символ.

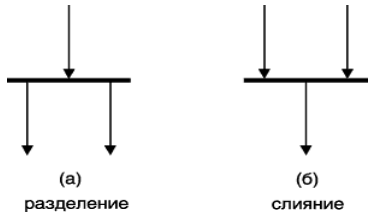


Рис. 62. Графическое изображение разделения и слияния параллельных потоков управления на диаграмме деятельности

На диаграммах деятельности такая черточка изображается отрезком горизонтальной, реже - вертикальной, линии, толщина которой несколько шире линий простых переходов диаграммы деятельности. При этом разделение (fork) имеет один входящий переход и несколько выходящих (рис. 62, а), которые изображаются отрезками вертикальных, реже - горизонтальных, линий. Слияние (join), наоборот, имеет несколько входящих переходов и один выходящий (рис. 62, б). Параллельные переходы на диаграмме деятельности можно изображать в удлиненной форме, а входящие и выходящие переходы вертикальными стрелками.

Пример: Регистрация пассажиров в аэропорту. Первоначально выполняется деятельность по проверке билета: если билет не действителен, он возвращается пассажиру; если билет действителен, то пассажиру выдается посадочный талон, в дополнение проверяется гражданство и наличие багажа у пассажира. Если есть багаж, то его проверка может быть выполнена параллельно, после чего пассажиру выдается талон на багаж. Если пассажир является иностранным гражданином, то дополнительно проверяется наличие у него визы. Если виза действительна, то проверка завершается успешно, и пассажир может проследовать на посадку. Если же виза не действительна, то для этого пассажира посадка оказывается невозможной, и ему не выдается посадочный талон и талон на багаж. Происходит прекращение всех выполняемых сотрудниками аэропорта действий.

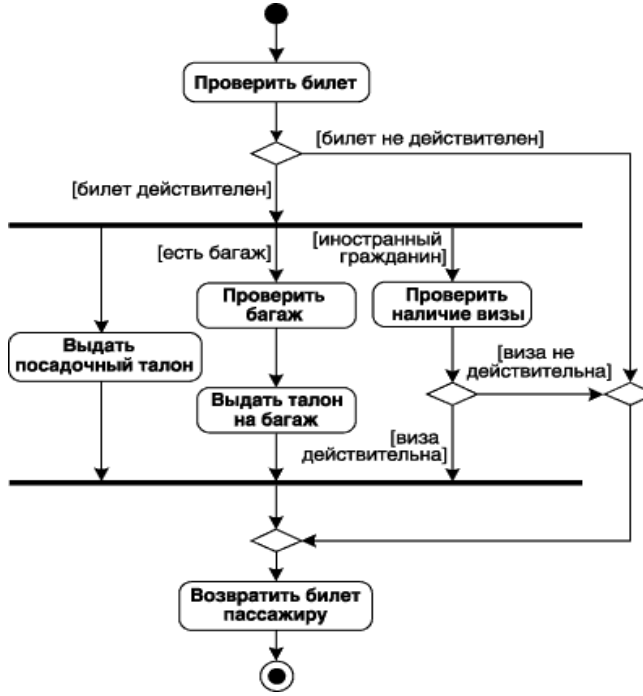


Рис. 63. Диаграмма деятельности для примера регистрации пассажиров в аэропорту

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах. Не менее важная область их применения связана с моделированием бизнес-процессов. В этом контексте деятельность любой компании или фирмы представляет собой не что иное, как совокупность отдельных действий, работ, операций, направленных на достижение требуемого результата.

Однако применительно к бизнес-процессам желательно выполнение каждого действия ассоциировать с конкретным подразделением компании. В этом случае подразделение будет нести ответственность за реализацию определенных действий, а сам бизнес-процесс представляется в виде переходов действий из одного подразделения к другому. Для моделирования этих особенностей в языке UML предложена специальная конструкция, получившая название дорожки.

Дорожка (swimlane) - графическая область диаграммы деятельности, содержащая элементы модели, ответственность за выполнение которых принадлежит отдельным подсистемам.

В данном случае имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму деятельности сверху. При этом все состояния на диаграмме деятельности делятся на группы, разграниченные вертикальными линиями.

Две соседних линии и образуют дорожку, а группа состояний между этими линиями выполняется организационным подразделением (отделом, группой, отделением, филиалом) или сотрудником компании (рис. 64). В последнем случае принято указывать должность сотрудника, ответственного за выполнение определенных действий.

Названия подразделений или должностей 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.

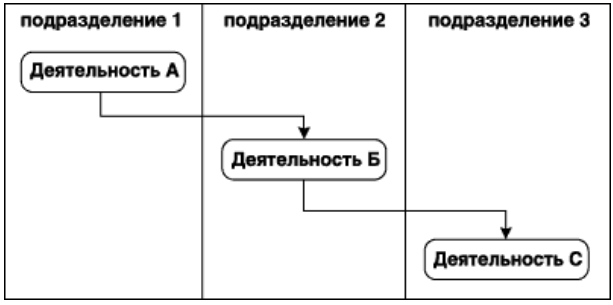


Рис. 64. Вариант диаграммы деятельности с дорожками

Пример: фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов в форме заказов. Подразделениями компании обычно являются отдел приема и оформления заказов, отдел продаж и склад. Этим подразделениям будут

соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения. В этом случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое подразделение торговой компании должно выполнять то или иное действие (рис. 65).

После принятия заказа от клиента отделом приема и оформления заказов осуществляется распараллеливание деятельности на два потока (переход-разделение). Первый из них остается в этом же отделе и связан с получением оплаты от клиента за заказанный товар. Второй инициирует выполнение действия по регистрации заказа в отделе продаж (модель товара, размеры, цвет, год выпуска и пр.). Однако выдача товара со склада начинается только после того, как будет получена от клиента оплата за товар (переход-слияние). Затем выполняется подготовка товара к отправке и его отправка клиенту в отделе продаж. После завершения этих деятельностей заказ закрывается в отделе приема и оформления заказов.

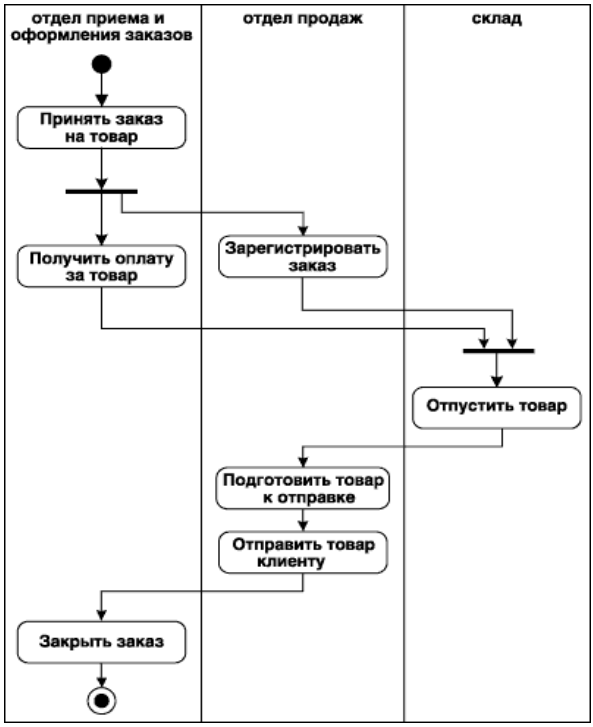


Рис. 65. Фрагмент диаграммы деятельности для торговой компании

Действия на диаграмме деятельности могут производиться над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности другому. Поскольку в таком ракурсе

объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме деятельности.

На диаграмме деятельности с дорожками расположение объекта может иметь дополнительный смысл. А именно, если объект расположен на границе двух дорожек, то это может означать, что переход к следующему состоянию действия в соседней дорожке ассоциирован с нахождением документа в некотором состоянии. Если же объект расположен внутри дорожки, то и состояние этого объекта целиком определяется действиями данной дорожки.

В примере с торговой компанией центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до обращения клиента заказ как объект отсутствует и возникает после контакта с клиентом. В результате фиксируется полученный заказ, потом он регистрируется в отделе продаж, затем он передается на склад, где после получения оплаты за товар оформляется окончательно. После того, как товар отправлен клиенту, эта информация вносится в заказ, и он считается выполненным (рис. 66).

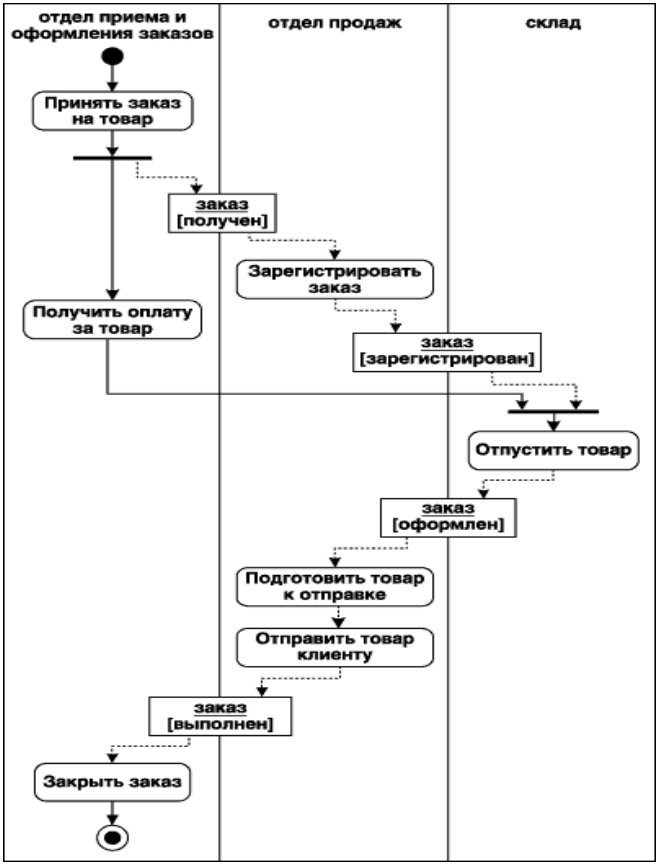


Рис. 66. Фрагмент диаграммы деятельности торговой компании с объектом-заказом

Достоинством диаграммы деятельности является возможность визуализировать отдельные аспекты поведения рассматриваемой системы или реализации отдельных операций классов в виде процедурной последовательности действий. Таким образом, полная модель системы может содержать одну или несколько диаграмм деятельности, каждая из которых описывает последовательность реализации либо наиболее важных вариантов использования (типичный ход событий и все исключения), либо нетривиальных операций классов.

Пример: Программное средство представляет собой базу данных «Автоматизация процесса составления расписания в учебном заведении». Программное средство обеспечивает корректировку данных, а именно в БД «Группы» может изменяться перечень предметов в соответствие с курсом группы и отделением; в БД «Предметы» могут изменяться номера аудиторий и фамилии преподавателей; осуществляет поиск по ФИО преподавателя, номеру аудитории, названию предмета и номеру группы. Программное средство составляет расписание работы для конкретного преподавателя на неделю; для группы на неделю; для группы по конкретному предмету, а также отчет о загрузке аудиторий на каждый день.

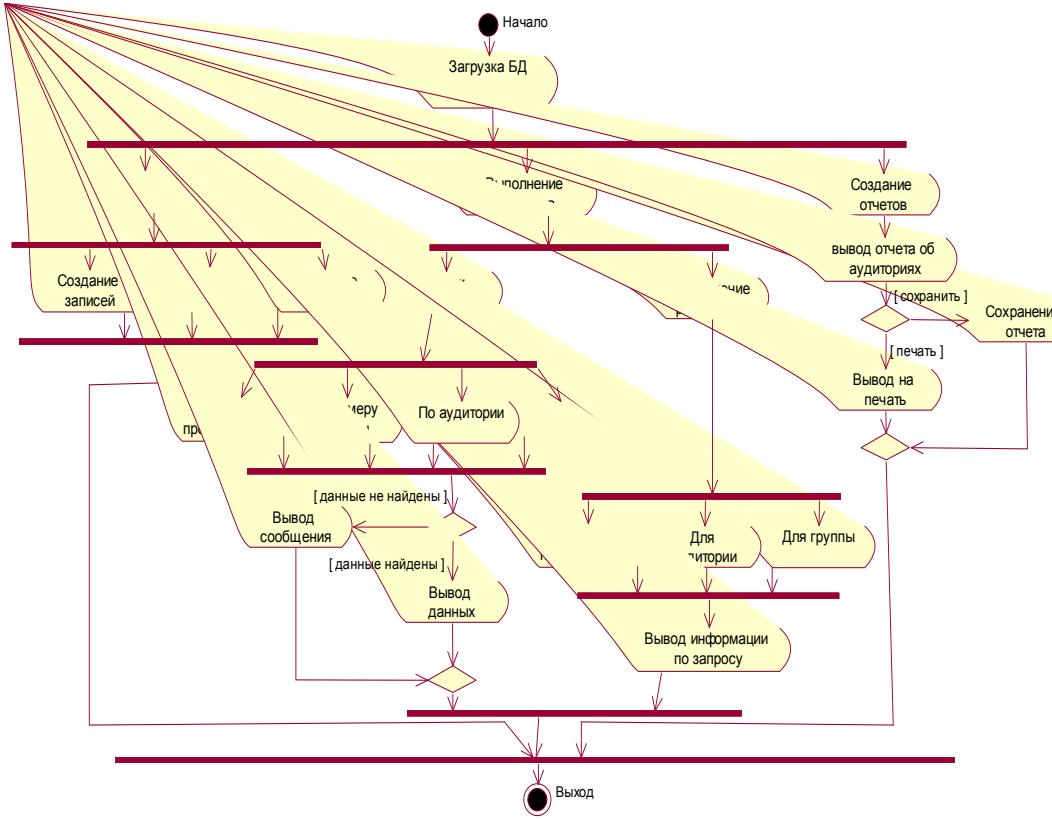


Рис. 67. Пример диаграммы деятельности

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Диаграмма деятельности ”.
2. Разработать диаграмму деятельности для произвольной системы индивидуального задания (Приложение А).
3. Оформить отчет, включив в него описание всех компонентов диаграммы деятельности согласно индивидуальному варианту задания, ответы на контрольные вопросы.

Контрольные вопросы

1. Дайте определение понятию «диаграмма деятельности».
2. Опишите назначение диаграммы деятельности.
3. Дайте определение понятиям «состояние деятельности» и «состояние действия». Графическое изображение состояния.
4. Приведите пример ветвления и параллельных потоков управления процессами на диаграмме деятельности.
5. Какие переходы используются на диаграмме деятельности?
6. Что представляет собой дорожка на диаграмме деятельности?
7. Как графически изображаются объекты на диаграмме деятельности?

ЛАБОРАТОРНАЯ РАБОТА № 7
ДИАГРАММА КОМПОНЕНТОВ

Цель работы: закрепление теоретических сведений о диаграмме компонентов; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме компонентов.

Краткие теоретические сведения

Для создания конкретной физической системы необходимо реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно – физическое представление модели. В контексте языка UML это означает совокупность связанных физических сущностей, включая программное и аппаратное обеспечение, а также персонал, которые организованы для выполнения специальных задач. Физическая система — реально существующий прототип модели системы.

Полный проект программной системы представляет собой совокупность моделей логического и физического представлений, которые должны быть согласованы между собой. В языке UML для физического представления моделей систем используются так называемые диаграммы реализации, которые включают в себя две отдельные канонические диаграммы: диаграмму компонентов и диаграмму развертывания.

Диаграмма компонентов (component diagram), в отличие от ранее рассмотрен-

ных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

В разработке диаграмм компонентов участвуют как системные аналитики и архитекторы, так и программисты. Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие – на этапе его исполнения. Диаграмма компонентов отражает общие зависимости между компонентами, рассматривая последние в качестве отношений между ними.

Для представления физических сущностей в языке UML применяется специальный термин – компонент.

Компонент (component) — физически существующая часть системы, которая обеспечивает реализацию классов и отношений, а также функционального поведения моделируемой программной системы.

Компонент предназначен для представления физической организации ассоциированных с ним элементов модели. Дополнительно компонент может иметь текстовый стереотип и помеченные значения, а некоторые компоненты – собственное графическое представление. Компонентом может быть исполняемый код отдельного модуля, командные файлы или файлы, содержащие интерпретируемые скрипты.

Компонент служит для общего обозначения элементов физического представления модели и может реализовывать некоторый набор интерфейсов. Для графического представления компонента используется специальный символ – прямоугольник со вставленными слева двумя более мелкими прямоугольниками (рис. 68). Внутри объемлющего прямоугольника записывается имя компонента и, возможно, дополнительная информация. Этот символ является базовым обозначением компонента в языке UML.

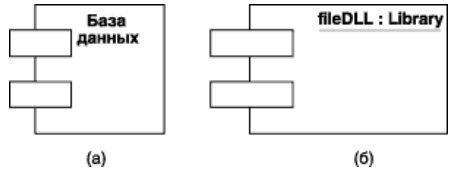


Рис. 68. Графическое изображение компонента

Графическое изображение компонента ведет свое происхождение от обозначения модуля программы, применявшегося некоторое время для отображения особенностей инкапсуляции данных и процедур.

Модуль (module) — часть программной системы, требующая памяти для своего хранения и процессора для исполнения.

В этом случае верхний маленький прямоугольник концептуально ассоциировался с данными, которые реализует этот компонент (иногда он изображается в форме овала). Нижний маленький прямоугольник ассоциировался с операциями или методами, реализуемыми компонентом. В простых случаях имена данных и методов записывались явно в маленьких прямоугольниках, однако в языке UML они не указываются.

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и знаков препинания. Отдельный компонент может быть представлен на уровне типа или экземпляра. И хотя его графическое изображение в обоих случаях одинаково, правила записи имени компонента несколько отличаются.

Если компонент представляется на уровне типа, то записывается только имя типа с заглавной буквы в форме: <Имя типа>. Если же компонент представляется на уровне экземпляра, то его имя записывается в форме: <имя компонента ':' Имя типа>. При этом вся строка имени подчеркивается. Так, в первом случае (рис. 68, а) для компонента уровня типов указывается имя типа, а во втором (рис. 68, б) для компонента уровня экземпляра – собственное имя компонента и имя типа.

Правила именования объектов в языке UML требуют подчеркивания имени отдельных экземпляров, но применительно к компонентам подчеркивание их имени часто опускают. В этом случае запись имени компонента со строчной буквы характеризует компонент уровня примеров.

В качестве собственных имен компонентов принято использовать имена исполняемых файлов, динамических библиотек, Web-страниц, текстовых файлов или файлов справки, файлов баз данных или файлов с исходными текстами программ, файлов скриптов и другие.

В отдельных случаях к простому имени компонента может быть добавлена информация об имени объемлющего пакета и о конкретной версии реализации данного компонента. Необходимо заметить, что в этом случае номер версии записывается как помеченное значение в фигурных скобках. В других случаях символ компонента может быть разделен на секции, чтобы явно указать имена реализованных в нем классов или интерфейсов. Такое обозначение компонента называется *расширенным*.

Поскольку компонент как элемент модели может иметь различную физическую реализацию, иногда его изображают в форме специального графического символа, иллюстрирующего конкретные особенности реализации. Строго говоря, эти дополнительные обозначения не специфицированы в нотации языка UML. Однако, удовлетворяя общим механизмам расширения языка UML, они упрощают понимание диаграммы компонентов, существенно повышая наглядность графического представления.

Для более наглядного изображения компонентов были предложены и стали общепринятыми следующие графические стереотипы:

– стереотипы для компонентов развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут

быть динамически подключаемые библиотеки (рис. 69, а), Web-страницы на языке разметки гипертекста (рис. 69, б) и файлы справки (рис. 69, в);

– стереотипы для компонентов в форме рабочих продуктов. Как правило – это файлы с исходными текстами программ (рис. 69, г).

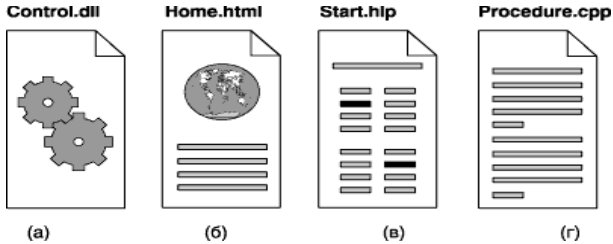


Рис. 69. Варианты графического изображения компонентов

Эти элементы иногда называют **артефактами**, подчеркивая при этом их законченное информационное содержание, зависящее от конкретной технологии реализации соответствующих компонентов. Более того, разработчики могут для этой цели использовать самостоятельные обозначения, поскольку в языке UML нет строгой нотации для графического представления артефактов.

Другой способ спецификации различных видов компонентов — указание текстового стереотипа компонента перед его именем. В языке UML для компонентов определены следующие стереотипы:

– <<file>> (файл) – определяет наиболее общую разновидность компонента, который представляется в виде произвольного физического файла;

– <<executable>> (исполнимый) – определяет разновидность компонента-файла, который является исполнимым файлом и может выполняться на компьютерной платформе;

– <<document>> (документ) – определяет разновидность компонента-файла, который представляется в форме документа произвольного содержания, не являющегося исполнимым файлом или файлом с исходным текстом программы;

– <<library>> (библиотека) – определяет разновидность компонента-файла, который представляется в форме динамической или статической библиотеки;

– <<source>> (источник) – определяет разновидность компонента-файла, представляющего собой файл с исходным текстом программы, который после компиляции может быть преобразован в исполнимый файл;

– <<table>> (таблица) – определяет разновидность компонента, который представляется в форме таблицы базы данных.

Следующим графическим элементом диаграммы компонентов являются интерфейсы. В общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (рис. 70, а). При этом имя интерфейса, которое рекомендуется начинать с заглавной буквы "I", записывается рядом с окружностью. Семантически линия означает реализацию интерфейса, а наличие интерфейсов у компонента означает, что данный компонент реализует соответствующий набор интерфейсов.

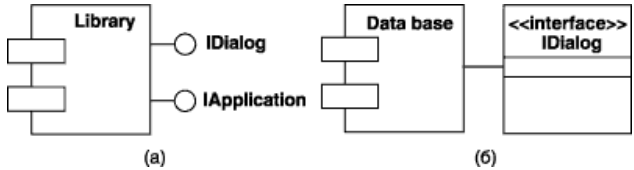


Рис. 70. Графическое изображение интерфейсов на диаграмме компонентов

Кроме того, интерфейс на диаграмме компонентов может быть изображен в виде прямоугольника класса со стереотипом <<interface>> и секцией поддерживаемых операций (рис. 70, б). Как правило, этот вариант обозначения используется для представления внутренней структуры интерфейса.

При разработке программных систем интерфейсы обеспечивают не только совместимость различных версий, но и возможность вносить существенные изменения в одни части программы, не изменяя другие. Характер применения интерфейсов отдельными компонентами может отличаться.

Различают два способа связи интерфейса и компонента. Если компонент реализует некоторый интерфейс, то такой интерфейс называют экспортируемым или *поддерживаемым*, поскольку этот компонент предоставляет его в качестве сервиса другим компонентам. Если же компонент использует некоторый интерфейс, который реализуется другим компонентом, то такой интерфейс для первого компонента называется *импортируемым*. Особенность импортируемого интерфейса состоит в том, что на диаграмме компонентов это отношение изображается с помощью зависимости.

Отношение зависимости служит для представления факта наличия специальной формы связи между двумя элементами модели, когда изменение одного элемента модели оказывает влияние или приводит к изменению другого элемента модели. Отношение зависимости на диаграмме компонентов изображается пунктирной линией со стрелкой, направленной от клиента или зависимого элемента к источнику или независимому элементу модели.

Зависимости могут отражать связи отдельных файлов программной системы на этапе компиляции и генерации объектного кода. В других случаях зависимость может указывать на наличие в независимом компоненте описаний классов, которые используются в зависимом компоненте для создания соответствующих объектов. Применительно к диаграмме компонентов зависимости могут связывать компоненты и импортируемые этим компонентом интерфейсы, а также различные виды компонентов между собой. В этом случае рисуют стрелку от компонента-клиента к импортируемому интерфейсу (рис. 71). Наличие такой стрелки означает, что компонент не реализует соответствующий интерфейс, а использует его в процессе своего выполнения. При этом на этой же диаграмме может присутствовать и другой компонент, который реализует этот интерфейс. **Отношение реализации** интерфейса обозначается на диаграмме компонентов обычной линией без стрелки.

Так, например, изображенный ниже фрагмент диаграммы компонентов представляет информацию о том, что компонент с именем «Control» зависит от импортируемого интерфейса «IDialog», который, в свою очередь, реализуется компонен-

том с именем «DataBase». При этом для второго компонента этот интерфейс является экспортируемым. Изобразить связь второго компонента «DataBase» с этим интерфейсом в форме зависимости нельзя, поскольку этот компонент реализует указанный интерфейс.

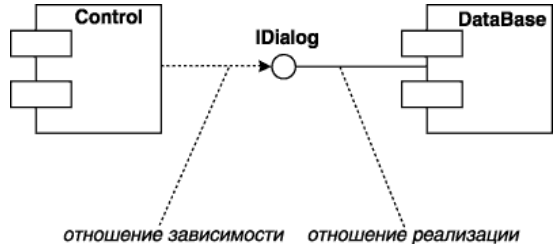


Рис. 71. Отношения зависимости и реализации между компонентами

Другим случаем отношения зависимости на диаграмме компонентов является **отношение программного вызова и компиляции** между различными видами компонентов.

Для рассмотренного фрагмента диаграммы компонентов (рис. 72) наличие подобной зависимости означает, что исполнимый компонент «Control.exe» использует или импортирует некоторую функциональность компонента «Library.dll», вызывает страницу гипертекста «Home.html» и файл помощи «Search.hlp», а исходный текст этого исполнимого компонента хранится в файле «Control.cpp». При этом характер отдельных видов зависимостей может быть отмечен дополнительно с помощью текстовых стереотипов.

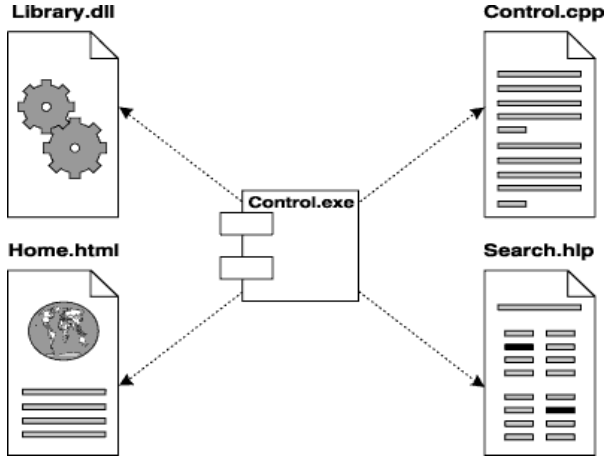


Рис. 72. Графическое изображение отношения зависимости между компонентами

На диаграмме компонентов могут быть также представлены отношения зависимости между компонентами и реализованными в них классами. Эта информация

имеет значение для обеспечения согласования логического и физического представлений модели системы. Изменения в структуре описаний классов могут привести к изменению этой зависимости. Ниже приводится фрагмент зависимости подобного рода, когда исполнимый компонент «Control.exe» зависит от соответствующих классов (рис. 73).

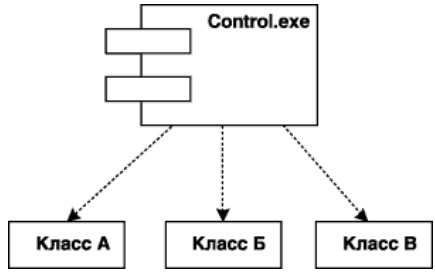


Рис. 73. Графическое изображение зависимости между компонентом и классами

В этом случае не следует, что классы реализованы данным компонентом. Если требуется подчеркнуть, что некоторый компонент реализует отдельные классы, то для его обозначения используется расширенный символ прямоугольника, который делится на две секции. Верхняя секция служит для записи имени компонента и дополнительной информации, а нижняя секция – для указания реализуемых данным компонентом классов (рис. 74).

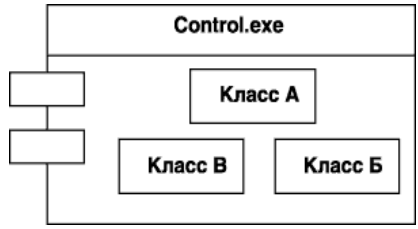


Рис. 74. Графическое изображение компонента с информацией о реализуемых им классах

В случае если компонент является экземпляром и реализует три отдельных объекта, он изображается в форме компонента уровня экземпляров (рис. 75). Объекты, которые находятся в отдельном компоненте-экземпляре, изображаются вложенными в данный компонент.

Подобная вложенность означает, что выполнение компонента влечет за собой выполнение операций соответствующих объектов. При этом существование компонента в течение времени исполнения программы обеспечивает функциональность всех вложенных в него объектов. Что касается доступа к этим объектам, то он может быть дополнительно специфицирован с помощью видимости, подобно видимости пакетов.

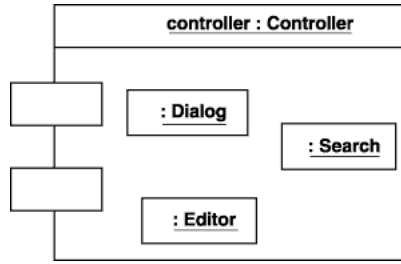


Рис. 75. Графическое изображение компонента-экземпляра, реализующего отдельные объекты

Для компонентов с исходным текстом программы видимость может означать возможность внесения изменений в соответствующие тексты программ с их последующей перекомпиляцией. Для компонентов с исполняемым кодом программы видимость может характеризовать возможность запуска на исполнение соответствующего компонента или вызова реализованных в нем операций или методов.

Разработка диаграммы компонентов предполагает использование информации не только о логическом представлении модели системы, но и об особенностях ее физической реализации. В первую очередь, необходимо решить, из каких физических частей или файлов будет состоять программная система. На этом этапе следует обратить внимание на такую реализацию системы, которая обеспечивала бы возможность повторного использования кода за счет рациональной декомпозиции компонентов, а также создание объектов только при их необходимости.

Общая производительность программной системы существенно зависит от рационального использования вычислительных ресурсов. Для этой цели необходимо большую часть описаний классов, их операций и методов вынести в динамические библиотеки, оставив в исполняемых компонентах только самые необходимые для инициализации программы фрагменты программного кода. После общей структуризации физического представления системы необходимо дополнить модель интерфейсами и схемами базы данных. При разработке интерфейсов следует обращать внимание на согласование различных частей программной системы. Включение в модель схемы базы данных предполагает спецификацию отдельных таблиц и установление информационных связей между ними.

Завершающий этап построения диаграммы компонентов связан с установлением и нанесением на диаграмму взаимосвязей между компонентами, а также отношений реализации. Эти отношения должны иллюстрировать все важнейшие аспекты физической реализации системы, начиная с особенностей компиляции исходных текстов программ и заканчивая исполнением отдельных частей программы на этапе ее выполнения. Для этой цели можно использовать различные графические стереотипы компонентов.

Если же проект содержит физические элементы, описание которых отсутствует в языке UML, то следует воспользоваться механизмом расширения. В частности, можно применить дополнительные стереотипы для отдельных нетиповых компонентов или помеченные значения для уточнения отдельных характеристик компо-

нентов.

Диаграммой развертывания, на которой представляется информация о физическом размещении компонентов программной системы по ее отдельным узлам.

Пример: Программное средство представляет собой среду для автоматизированного построения графических объектов в 3D-пространстве. Оно обрабатывает входные данные, вводимые пользователем, обеспечивает их визуализацию, а также предоставляет возможность различных преобразований над объектами, вариантами их отображения. Программа позволяет строить различные графические объекты, как окружность, куб, гиперболоид, параболоид, различные двумерные и трехмерные графики. Также программа предоставляет возможность управления опциями, имеет файл справки и помощи.

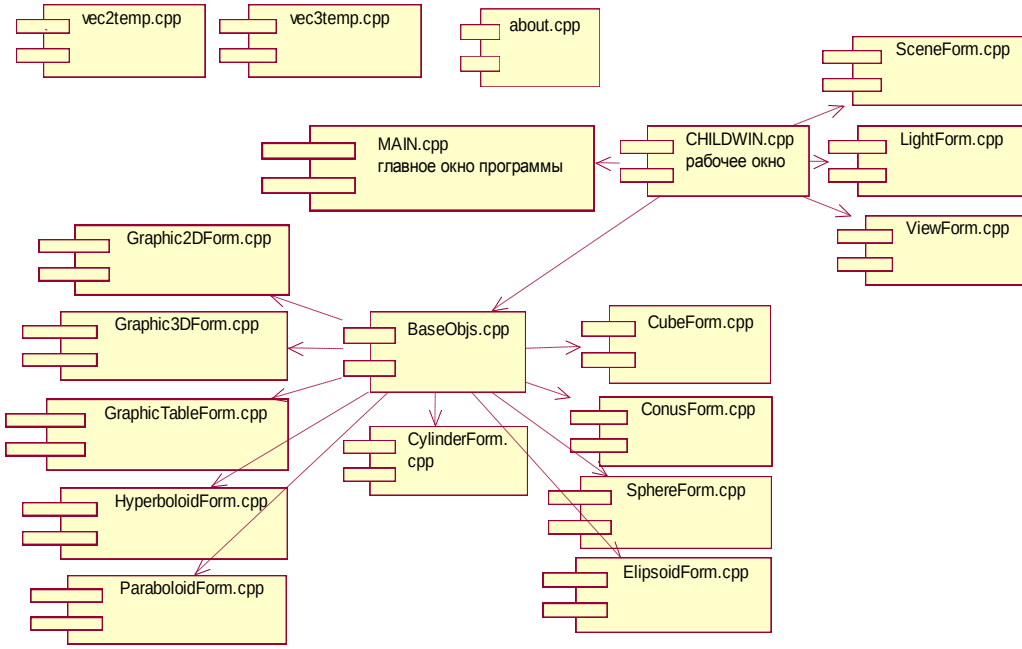


Рис. 76. Пример диаграммы компонентов

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Диаграмма компонентов”.
2. Разработать диаграмму компонентов для произвольной системы индивидуального задания (Приложение А).
3. Оформить отчет, включив в него описание всех компонентов диаграммы компонентов согласно индивидуальному варианту задания, ответы на контрольные вопросы.

Контрольные вопросы

1. Дайте определение понятию «диаграмма компонентов» и опишите назначение диаграммы компонентов.
2. Дайте определение понятиям «компонент», «модуль». Графическое изображение компонента и модуля.
3. Какие стереотипы определены для компонентов?
4. Что представляет собой интерфейс на диаграмме компонентов? Назовите способы связи компонента и интерфейса.
5. Какие вы знаете способы связи между компонентами? Приведите примеры.

ЛАБОРАТОРНАЯ РАБОТА № 8
ДИАГРАММА РАЗВЕРТЫВАНИЯ

Цель работы: закрепление теоретических сведений о диаграмме развертывания; овладение практическими навыками моделирования процессов, описывающих взаимодействие объектов в диаграмме развертывания.

Краткие теоретические сведения

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Если создается простая программа, которая может выполняться локально на компьютере пользователя, не используя никаких распределенных устройств и сетевых ресурсов, то необходимости в разработке дополнительных диаграмм нет. При создании корпоративных или распределенных приложений требуется визуализировать сетевую инфраструктуру программной системы.

Сложные программные системы могут реализовываться в сетевом варианте, на различных вычислительных платформах и технологиях доступа к распределенным базам данных. Наличие локальной корпоративной сети требует решения целого комплекса дополнительных задач рационального размещения компонентов по узлам этой сети, что определяет общую производительность программной системы.

Интеграция программной системы с Интернетом определяет необходимость решения дополнительных вопросов при проектировании системы (обеспечение безопасности и устойчивости доступа к информации). Эти аспекты зависят от реализации проекта в форме физически существующих узлов системы, таких как серверы, рабочие станции, брандмауэры, каналы связи и хранилища данных.

Технологии доступа и манипулирования данными в рамках общей схемы "клиент-сервер" также требуют размещения больших баз данных в различных сегментах корпоративной сети, их резервного копирования, архивирования, кэширования для обеспечения необходимой производительности системы в целом. С целью спецификации программных и технологических особенностей реализации распределенных архитектур необходимо визуальное представление этих аспектов.

Диаграмма развертывания (deployment diagram) - диаграмма, на которой представлены узлы выполнения программных компонентов реального времени, а также процессов и объектов.

Диаграмма развертывания применяется для представления общей конфигурации и топологии распределенной программной системы и содержит изображение размещения компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений - маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих только на этапе ее исполнения (run-time). При этом представляются только те компоненты программы, которые являются исполнимыми файлами или динамическими библиотеками. Компоненты, не используемые на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единственной для системы в целом, поскольку должна отражать все особенности ее реализации.

Узел (node) представляет собой физически существующий элемент системы, который может обладать вычислительным ресурсом или являться техническим устройством.

В качестве вычислительного ресурса узла может рассматриваться один или несколько процессоров, а также объем электронной или магнитооптической памяти. Однако в языке UML понятие узла включает в себя не только вычислительные устройства (процессоры), но и другие механические или электронные устройства, такие как датчики, принтеры, модемы, цифровые камеры, сканеры и манипуляторы.

Графически узел на диаграмме развертывания изображается в форме трехмерного куба. Узел имеет имя, которое указывается внутри этого графического символа. Сами узлы могут представляться как на уровне типа (рис. 77, а), так и на уровне экземпляра (рис. 77, б).

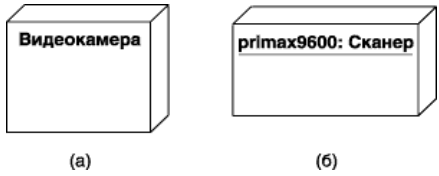


Рис. 77. Графическое изображение узла на диаграмме развертывания

В первом случае имя узла записывается в форме: <Имя типа узла> без подчеркивания и начинается с заглавной буквы. Во втором - имя узла-экземпляра записывается в виде: <имя узла ':' Имя типа узла>, а вся запись подчеркивается. Имя типа узла указывает на разновидность узлов, присутствующих в модели системы. Так, на представленном рисунке (рис. 77, а) узел с именем Видеокамера относится к об-

щему типу и никак не конкретизируется. Второй узел (рис. 77, б) является узлом-экземпляром конкретной модели сканера.

Изображения узлов могут расширяться, чтобы включить дополнительную информацию о спецификации узла. Если дополнительная информация относится к имени узла, то она записывается под именем в форме помеченного значения (рис. 78).

Рис. 78. Графическое изображение узла-экземпляра с дополнительной информацией в форме помеченного значения

При необходимости явно указать компоненты, которые размещаются или выполняются на отдельном узле, это можно сделать двумя способами. Первый из них позволяет разделить графический символ узла на две секции горизонтальной линией. В верхней секции записывают имя узла, а в нижней - размещенные на этом узле компоненты (рис. 79, а).

Второй способ разрешает показывать на диаграмме развертывания узлы с вложенными изображениями компонентов (рис. 79, б). Важно помнить, что в качестве таких вложенных компонентов могут выступать только исполняемые компоненты и динамические библиотеки.

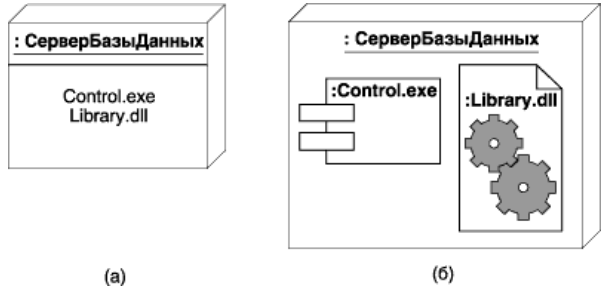


Рис. 79. Варианты графического изображения узлов-экземпляров с размещаемыми на них компонентами

В качестве дополнения к имени узла могут использоваться различные текстовые стереотипы, которые явно специфицируют назначение этого узла. Для этой цели были предложены следующие текстовые стереотипы: "processor" (процессор), "sensor" (датчик), "modem" (модем), "net" (сеть), "printer" (принтер) и другие, смысл которых понятен из контекста.

На диаграммах развертывания допускаются специальные условные обозначения

для различных физических устройств, графическое изображение которых проясняет назначение или выполняемые устройством функции. Однако пользоваться этой возможностью следует осторожно, памятуя о том, что основное достоинство языка UML следует из его названия - унификация графических элементов визуализации моделей.

Возможность включения персонала в понятие узла не рассматривается в нотации языка UML, тем не менее, подобное расширение понятие узла позволяет создавать средствами языка UML модели самых различных систем, включая бизнес-процессы и технические комплексы. Действительно, для реализации бизнес-процессов компаний удобно рассматривать в качестве узлов-ресурсов системы организационные подразделения, состоящие из персонала. Автоматизация управления техническими комплексами может потребовать рассмотрения в качестве самостоятельного элемента человека-оператора, способного принимать решения в нештатных ситуациях и нести ответственность за возможные последствия этих решений.

Наиболее известны два специальных графических стереотипа для обозначения разновидностей узлов. Первый обозначает ресурсоемкий узел (processor), под которым понимается узел с процессором и памятью, необходимыми для выполнения исполняемых компонентов. Он изображается в форме куба с боковыми гранями, окрашенными в серый цвет (рис. 80, а). Второй стереотип в форме обычного куба обозначает устройство (device), под которым понимается узел без процессора и памяти (рис. 80, б). На этом типе узлов не могут размещаться исполняемые компоненты программной системы.

Рис. 80. Варианты изображения графических стереотипов узлов

Кроме графического изображения ресурсоемких узлов и устройств соответствующие узлы можно изображать с помощью обычного символа узла (рис.80) и дополнительного стереотипа "processor" или "device".

Кроме известных текстовых и графических стереотипов для узлов диаграммы развертывания разработчики могут предложить дополнительные графические стереотипы, которые улучшают наглядность представления диаграмм развертывания. Например, рабочую станцию можно изобразить в виде ресурсоемкого узла, или в форме рисунка внешнего вида компьютера (рис. 80, в). Соответственно, сканер также может быть изображен в виде рисунка или фотографии данного устройства.

На диаграмме развертывания кроме изображения узлов указываются отношения между ними. В качестве отношений выступают физические соединения между узлами, а также зависимости между узлами и компонентами, которые допускается изображать на диаграммах развертывания.

Соединения являются разновидностью ассоциации и изображаются отрезками линий без стрелок. Наличие такой линии указывает на необходимость организации физического канала для обмена информацией между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, стереотипом, помеченным значением или ограничением. Так, на представленном ниже фрагменте диаграммы развертывания (рис. 81) явно определены рекомендации по технологии физической реализации соединений в форме примечания.

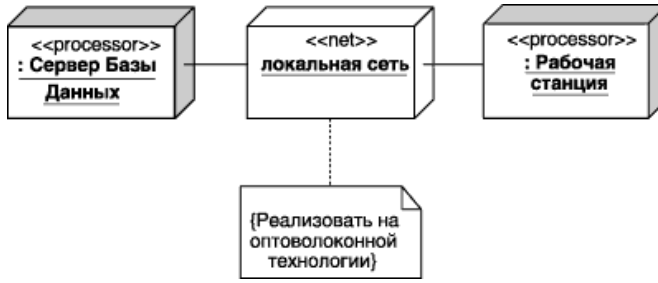


Рис. 81. Фрагмент диаграммы развертывания с соединениями между узлами

Кроме соединений на диаграмме развертывания могут присутствовать отношения зависимости между узлом и размещаемыми на нем компонентами. Подобный способ представляет собой альтернативу вложенному изображению компонентов внутри символа узла, что не всегда удобно, поскольку делает этот символ излишне объемным. При большом количестве развернутых на узле компонентов соответствующую информацию можно представить в форме отношения зависимости (рис. 82).

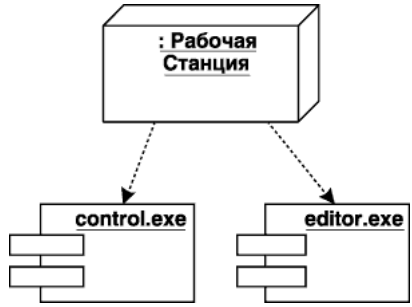


Рис. 82. Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами

Разработка информационных систем, обеспечивающих доступ в режиме реального времени, предполагает не только создание программного кода, но и использование дополнительных аппаратных средств. Вариант физического представления модели мобильного доступа к корпоративной базе данных показан на следующей диаграмме развертывания.

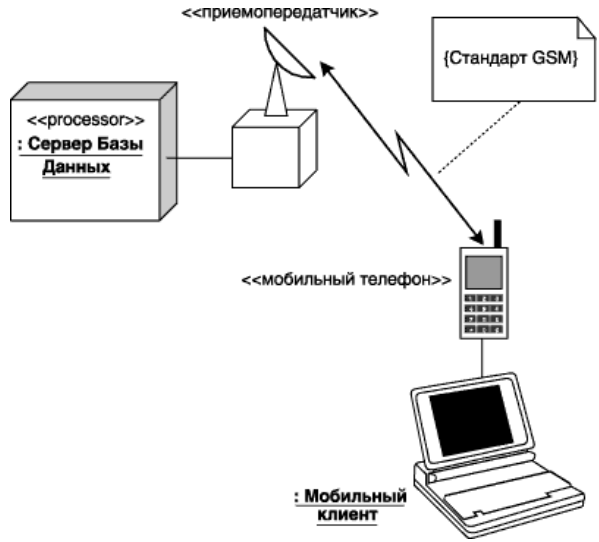


Рис. 83. Диаграмма развертывания для системы мобильного доступа к корпоративной базе данных

Данная диаграмма содержит общую информацию о развертывании рассматриваемой системы и может быть детализирована при разработке программных компонентов управления. Как видно из рисунка, в этой диаграмме развертывания использованы дополнительные стереотипы "приемопередатчик" и "мобильный телефон", которые отсутствуют в описании языка UML, а также специальные графические изображения (стереотипы) для отдельных аппаратных устройств.

Разработка диаграммы развертывания начинается с идентификации всех аппаратных, механических и других типов устройств, которые необходимы для выполнения системой всех функций. Вначале специфицируются вычислительные узлы системы, обладающие процессором и памятью. При этом используются имеющиеся в языке UML стереотипы, а, в случае отсутствия последних, разработчики могут определить новые стереотипы. Отдельные требования к составу аппаратных средств могут быть заданы в форме ограничений и помеченных значений.

Дальнейшая детализация диаграммы развертывания связана с размещением всех исполняемых компонентов диаграммы по узлам системы. В модели должна быть исключена ситуация, когда отдельные исполняемые компоненты оказались не размещенными на узлах. С этой целью можно внести в модель дополнительные узлы, содержащие процессор и память.

При разработке простых программ, которые исполняются локально на одном компьютере, необходимости в диаграмме развертывания нет. В более сложных ситуациях диаграмма развертывания строится для следующих приложений.

Во-первых, для моделирования программных систем, реализующих технологию доступа к данным "клиент-сервер". Для подобных систем характерно четкое разде-

ление полномочий и, соответственно, компонентов между клиентскими рабочими станциями и сервером базы данных. Возможность реализации "тонких" клиентов на простых терминалах или организация доступа к хранилищам данных приводит к необходимости уточнения не только топологии системы, но и ее компонентного состава.

Во-вторых, для моделирования неоднородных распределенных архитектур. Речь идет о корпоративных интрасетях, насчитывающих сотни компьютеров и других периферийных устройств, функционирующих на различных платформах и под различными операционными системами. При этом отдельные узлы такой системы могут быть удалены друг от друга на сотни километров, как, например, центральный офис в столице и филиалы компаний в регионах. В этом случае диаграмма развертывания становится важным инструментом визуализации общей топологии системы и контроля миграции отдельных компонентов между узлами.

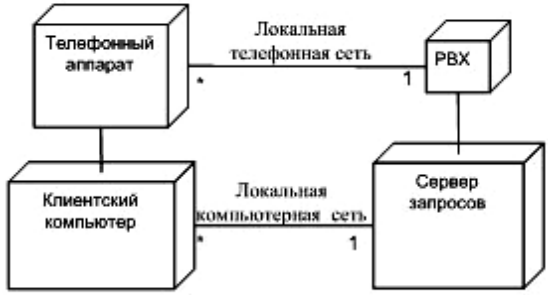
В-третьих, для моделирования систем реального времени со встроенными микропроцессорами, которые могут функционировать автономно. Такие системы могут содержать самые разнообразные дополнительные устройства, обеспечивающие автономность их функционирования при решении сложных целевых задач. Для подобных систем диаграмма развертывания позволяет визуализировать состав всех устройств и их взаимосвязи в системе.

Разработка диаграммы развертывания осуществляется на завершающем этапе ООАП, что характеризует окончание фазы проектирования физического представления.

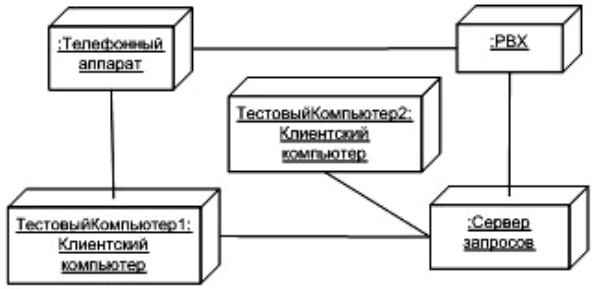
С другой стороны, диаграмма развертывания может строиться для анализа существующей системы с целью ее последующей детализации и модификации. При этом разработка диаграммы на начальных этапах анализа, характеризует его общее направление от физического представления к логическому.

Пример: На рис. 84., а показано, что телефонная служба приема заявок будет состоять из офисной телефонной станции (PBX - Public Branch Exchange), сервера, телефонных аппаратов и клиентских компьютеров. На этом рисунке представлена диаграмма развертывания в одном из двух возможных в UML видов - в описательном. На ней определены типы аппаратных узлов системы, а между ними - ассоциации с пометками множественности.

На рис. 84., б приведена диаграмма развертывания в экземплярном варианте. Показан тестовый вариант системы, который, кроме сервера и PBX, содержит один пользовательский компьютер для тестирования взаимодействия сервера и клиента и один клиентский компьютер вместе с телефонным аппаратом для тестирования связи клиента с сервером и PBX. Два клиентских компьютера нужны, чтобы тестировать работу ПО в случае более чем одного клиента (при переходе от одного к двум начинают появляться многочисленные ситуации, которые не проявлялись ранее). Большее количество клиентов - три, десять и т. д. - не принципиально на первых стадиях тестирования и отладки.



а). Диаграмма развертывания: описательный уровень



б). Диаграмма развертывания: экземплярный уровень

Рис. 84. Пример диаграммы развертывания

Порядок выполнения работы

1. Изучить теоретические сведения по теме “Диаграмма развертывания”.
2. Разработать диаграмму развертывания для произвольной системы индивидуального задания (Приложение А).
3. Оформить отчет, включив в него описание всех компонентов диаграммы развертывания согласно индивидуальному варианту задания, ответы на контрольные вопросы.

Контрольные вопросы

1. Дайте определение понятию «диаграмма развертывания» и опишите назначение диаграммы развертывания.
2. Дайте определение понятию «узел». Графическое изображение узла.
3. Назовите основные стереотипы узлов.
4. Перечислите виды связей между узлами.

ПРИЛОЖЕНИЕ А. ВАРИАНТЫ ИНДИВИДУАЛЬНЫХ ЗАДАНИЙ

Для каждого из предложенных ниже вариантов требуется при помощи Visio или Visual построить модель программного обеспечения с использованием объектного подхода. Модель программного обеспечения включает в себя следующие канонические диаграммы:

- 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).

Все канонические диаграммы модели программного обеспечения должны удовлетворять перечисленным ниже требованиям:

- каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области;
- все сущности на диаграмме модели должны быть одного уровня представления;
- вся информация о сущностях должна быть явно представлена на диаграммах;
- диаграммы не должны содержать противоречивой информации;
- каждая диаграмма должна быть самодостаточной для правильной интерпретации всех ее элементов и понимания семантики всех используемых графических символов;
- любая модель системы должна содержать только те элементы, которые определены в нотации языка UML.

На диаграммах модели программного обеспечения все соответствующие компоненты должны сопровождаться описанием. Эти описания должны быть емкими и составляться на русском языке. Описание некоторых компонентов диаграмм должно включать в себя пояснение, предусловие, потоки событий (если таковые есть) и постусловие. Помимо описания компонентов на канонических диаграммах должны быть явно указаны стереотипы связей и сообщений между различными компонентами.

Все канонические диаграммы модели программного обеспечения должны быть оформлены в виде одного общего отчета в той последовательности, в которой они приведены в лабораторном практикуме, либо в виде отдельных отчетов по каждой из лабораторных работ.

Если диаграммы оформлены в виде общего отчета, то он должен включать в себя:

- условие индивидуального задания в соответствие с вариантом, выданным преподавателем;
- глоссарий проекта, оформленный в виде таблицы;

- пояснения каждой канонической диаграммы модели и ее графическое представление (скриншот) исходя из условия индивидуального задания;
- выводы по готовому проекту.

Отчет оформляется в Microsoft Word или в любом другом текстовом редакторе и предоставляется преподавателю в печатном виде после окончания выполнения данной лабораторной работы для отдельных отчетов, либо по окончанию всех лабораторных работ для общего отчета для его защиты.

Темы

1. Система автоматизации дли пункта проката видеокассет
2. Библиотека
3. Интернет-магазин
4. WWW-конференция
5. Будильник
6. Генеалогическое дерево (программа)
7. Поддержки составления расписания занятий
8. Поликлиника
9. Парикмахерской
10. Система управления учебным процессом
11. Система "Клиент-Банк"
12. Универсальная система "Склад"
13. Система столовой
14. Автозаправка
15. Аптека
16. Швейное ателье
17. Газетный киоск
18. Оформления подписки на почте
19. Аэропорта
20. Речного вокзала
21. Железнодорожного вокзал
22. Автовокзала
23. Троллейбусное дэпо
24. Таксопарк
25. Книжный магазин
26. Съемки фильма
27. Съемки мультфильма
28. Компания «Три слона», занимающаяся грузоперевозками по городу
29. Компания «Синдбад», занимающаяся перевозкой груза на

большие расстояния.

30. Лаборатории, занимающейся селекцией лис для выведения ручных зверей.
31. Отделения кинологов, занимающихся дрессировкой и разведением собак.

1.

ЛИТЕРАТУРА

1. Вендров А.М. Объектно-ориентированный анализ и проектирование с использованием языка UML и Rational Rose.
2. Грейди Буч, Джеймс Рамбо, Айвар Джекобсон. UML. Руководство пользователя.
3. Джим Арлоу, Айла Нейштадт. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. – Символ-Плюс, 2007. – 624с.
4. Хассан Гома. UML-проектирование систем реального времени параллельных и распределенных приложений. – ДМК Пресс, 2011 – 704с.
5. http://www.intuit.ru –учебный курс «Нотация и семантика языка UML» А.В. Леоненков