Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»

Кафедра информационных систем и информационного менеджмента

ЯЗЫК ВИЗУАЛЬНОГО МОДЕЛИРОВАНИЯ UML

Методические указания к курсовой работе
по дисциплине «Разработка и стандартизация программных
средств и технологий»

Составители
А. В. КОНУШИН
В. И. МАЗАНОВА

Владимир 2012

УДК 004.4
ББК 32.973
Я41

Рецензент
Кандидат технических наук,
доцент кафедры информационных систем
и информационного менеджмента
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых
В.В. Вершинин

Печатается по решению редакционно-издательского совета ВлГУ

Я41 **Язык** визуального моделирования UML : метод. указания к курсовой работе по дисциплине «Разработка и стандартизация программных средств и технологий» / Владим. гос. ун-т имени Александра Григорьевича и Николая Григорьевича Столетовых ; сост. : А. В. Конушин, В. И. Мазанова. – Владимир : Изд-во ВлГУ, 2012. – 31 с.

Содержат описание требований к структуре и содержанию курсовой работы по дисциплине «Разработка и стандартизация программных средств и технологий». Включают краткие теоретические сведения по рассматриваемым вопросам, порядок выполнения курсовой работы, примерный перечень тем.

Направлены на изучение и освоение подходов к анализу и проектированию программных систем с использованием языка UML и программного средства Microsoft Visio.

Предназначены для студентов 3-го курса очной формы обучения специальности 080801 – прикладная информатика в экономике и других, изучающих дисциплины соответствующего профиля.

Рекомендованы для формирования профессиональных компетенций в соответствии с ФГОС 3-го поколения.

Ил. 32. Библиогр.: 5 назв.

УДК 004.4
ББК 32.973

ВВЕДЕНИЕ

В основе любой отрасли промышленного производства, к которым относится и создание программного обеспечения (ПО) или программных средств (ПС), лежит технологический процесс. Большинство характеристик программного продукта – качество, стоимость, сроки создания, актуальность – непосредственно определяются технологией разработки и точностью ее соблюдения.

Принятый консорциумом Object Management Group в 1997 году в качестве стандарта, унифицированный язык моделирования (UML) быстро распространился в сфере производства ПО как графический язык для спецификирования, создания, визуализации и документирования систем, в которых большая роль принадлежит программному обеспечению. В настоящее время он поддерживается многими объектно-ориентированными средствами разработки.

UML позволяет охватить не только концептуальные элементы системы – бизнес-процессы, системные функции, но и конкретные детали: классы языков программирования, схемы баз данных, повторно используемые компоненты программного обеспечения.

.

ПРЕДИСЛОВИЕ

Настоящие методические указания разработаны для выполнения курсовой работы (КР), посвящённой проектированию программных систем с использованием визуального языка моделирования (UML). В ходе работы у студента должно сформироваться четкое представление о современных подходах к проектированию и разработке программных систем с использованием нотации UML и соответствующих CASE-средств. Представленные варианты индивидуальных заданий это позволят изучить все особенности разноуровневого подхода к моделированию программной системы и выполнить курсовую работу в рамках изучения дисциплины «Разработка и стандартизация программных средств и технологий».

Общее задание для выполнения курсовой работы

1. Исследовать выбранную предметную область.
2. Изучить графическую нотацию визуального языка моделирования UML 2.0.
3. Разработать визуальный проект рассматриваемой программной системы согласно требованиям нотации UML 2.0 и с использованием среды моделирования Microsoft Visio:
    3.1. Разработать концептуальную модель программной системы.
    3.2. Разработать логическую модель программной системы.
    3.3. Разработать динамическую модель программной системы.
    3.4. Разработать физическую модель программной системы
4. Оформить пояснительную записку.

Текстовая часть курсовой работы оформляется в соответствии с ГОСТ 7.32-91. При оформлении пояснительной записки учитываются также требования стандарта предприятия СТП 71.2-01, а также ГОСТ 2.105-95, ГОСТ 2.106-95 и другие стандарты, рекомендованные выпускающей кафедрой. Каждый раздел текстовой части КР должен начинаться с нового листа.

Объем курсовой работы составляет 30 – 50 стр.

Структура курсовой работы

1. Титульный лист.
2. Задание.
3. Аннотация *(1 стр.)*
4. Введение (постановка задачи; актуальность и практическая значимость выбранной предметной области; описание и сравнение аналогов проектируемой программной системы (≈ 2 стр.).
5. Проектирование ПС *(15 - 20 стр.):*

- концептуальная модель ПС;
- логическая модель ПС;
- динамическое представление логической модели ПС;
- физическая модель ПС.

6. Заключение (выводы, практическая ценность, возможности дальнейшей доработки или развития ПС) (≈ *2 стр)*.
7. Список использованных источников.

8. Приложения (сюда следует включать вспомогательные материалы, для того чтобы они не загружали текст пояснительной записки).

Тематика курсовых работ

1. Информационная система (ИС) медицинского учреждения.
2. ИС ресторана.
3. Подсистема «Библиотека» ИС «ВУЗ».
4. Подсистема «Деканат» ИС «ВУЗ».
5. Подсистема «Кафедра» ИС «ВУЗ».
6. Подсистема «Абитуриент» ИС «ВУЗ».
7. ИС гостиничного комплекса.
8. ИС сети магазинов.
9. ИС диспетчерской службы вокзала.
10. ИС он-лайн продаж (Интернет-магазин).
11. ИС отдела кадров завода по производству стекла.
12. ИС отдела технического контроля завода по производству стекла.
13. ИС отдела кредитования банковского учреждения.
14. Почтовый клиент.
15. Система учета рабочего времени.

ПРОЕКТИРОВАНИЕ ПС. ПОСТРОЕНИЕ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ ПРОГРАММНОЙ СИСТЕМЫ

Общие сведения

Диаграмма вариантов использования (use case diagram) описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением, или концептуальной моделью системы, в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует цели:

- определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
- сформулировать общие требования к функциональному поведению проектируемой системы;

- разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
- подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей, или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом *актером (actor),* или действующим лицом, называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, *вариант использования (use case)* служит для описания сервисов, которые система предоставляет актеру (рис.1). Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

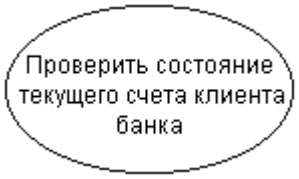

Рис. 1. Графическое обозначение варианта использования

Каждый вариант использования соответствует отдельному сервису, который предоставляет моделируемую сущность или систему по запросу пользователя (актера), т.е. определяет способ применения этой сущности. Сервис, который инициализируется по запросу пользователя, представляет собой законченную последовательность действий. Это означает, что после того как система закончит обработку запроса пользователя, она должна возвратиться в исходное состояние, в котором готова к выполнению следующих запросов.

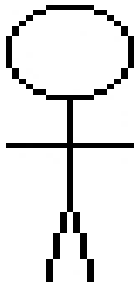

Рис. 2. Графическое обозначение актера

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования. Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается конкретное имя актера (рис. 2).

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

- отношение ассоциации, или association relationship (рис.3);
- отношение обобщения, или generalization relationship (рис. 4 –5);
- отношение расширения, или extend relationship (рис. 6);
- отношение включения, или include relationship (рис. 7).

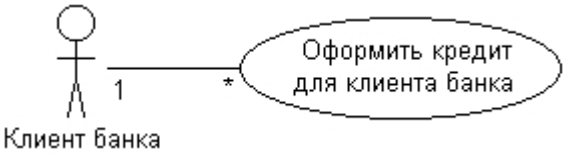

Рис. 3. Пример графического представления отношения ассоциации между актером и вариантом использования

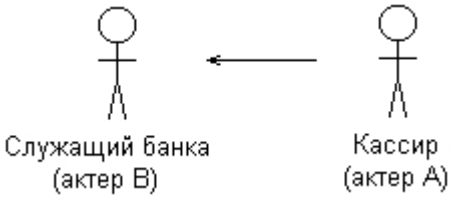

Рис. 4. Пример графического изображения отношения обобщения между актерами

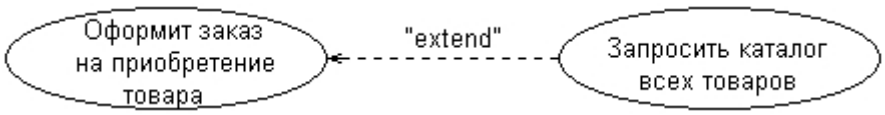

Рис. 4. Пример графического изображения отношения расширения между вариантами использования

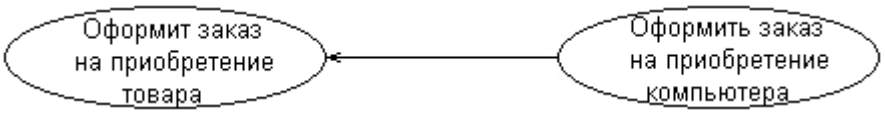

Рис. 5. Пример графического изображения отношения обобщения между вариантами использования

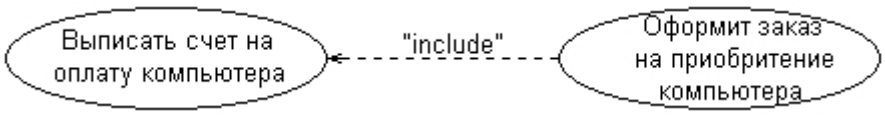

Рис. 7. Пример графического изображения отношения включения между вариантами использования

Шаблон описания варианта использования

1. ***Название*** варианта использования.

2. ***Введение*** *(*общее описание назначения данного варианта использования).

3. ***Предусловия*** (условия начала выполнения варианта использования).

4. *Основной поток событий (*последовательность событий и действий, приводящих к основной цели варианта использования).

5. *Альтернативный поток событий* (последовательности событий и действий, отклоняющихся от основного потока событий (например*, исключения, обработка ошибок, отказ в доступе и т.п.*).

6. *Постусловия* (состояние системы после выполнения варианта использования).

Задание

Выбрать предметную область из предложенных, либо предложить самостоятельно и согласовать с преподавателем. Проанализировать предметную область и выявить возможные функции для автоматизации их программной системой. Построить диаграмму вариантов использования для проектируемой системы. Уточнить диаграмму. Описать 3 – 4 варианта использования по приведенному шаблону.

Порядок выполнения работы.

1. В Microsoft Office Visio создать новую диаграмму. В качестве типа диаграммы выбрать Software - UML Model Diagram.
2. Используя только фигуры с закладки UML Use Case, построить диаграмму вариантов использования.
3. В любом текстовом редакторе описать 3 – 4 варианта использования по приведенному шаблону.

ПРОЕКТИРОВАНИЕ ПС. ПОСТРОЕНИЕ ЛОГИЧЕСКОЙ МОДЕЛИ ПРОГРАММНОЙ СИСТЕМЫ

Общие сведения

Центральное место в объектно-ориентированном анализе и проектировании (ООАП) занимает разработка логической модели системы в виде диаграммы классов. Нотация классов в языке UML проста и интуитивно понятна всем, кто когда-либо имел опыт работы с CASE-инструментариями.

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-

ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 8). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

Рис. 8. Графическое изображение класса на диаграмме классов: а – атрибуты; б – методы; в – исключения

Обязательным элементом обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса. По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами и операциями.

Даже если секция атрибутов или операций пустая, в обозначении класса она выделяется горизонтальной линией, для того чтобы сразу отличить класс от других элементов языка UML. Для класса "Счет" (рис. 8, *в*) дополнительно изображена четвертая секция, в которой указано исключение – отказ от обработки просроченной кредитной карточки.

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). В дополнение к общему правилу наименования элементов языка UML имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется

в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов. Необходимо помнить, что именно имена классов образуют словарь предметной области при ООАП.

Пример имен классов могут быть такие существительные, как "Сотрудник", "Компания", "Руководитель", "Клиент", "Продавец", "Менеджер", "Офис" и многие другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Во второй сверху секции прямоугольника класса записываются его *атрибуты (attributes),* или свойства. В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому атрибуту класса соответствует отдельная строка текста, состоящая из квантора видимости атрибута, имени атрибута, его кратности, типа значений атрибута и, возможно, его исходного значения:

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

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

- символ "+" обозначает атрибут с областью видимости типа «общедоступный» (public);
- символ "#" обозначает атрибут с областью видимости типа «защищенный» (protected);
- знак "-" обозначает атрибут с областью видимости типа «закрытый» (private).

Квантор видимости может быть опущен. Имя атрибута представляет собой строку текста, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса. Имя является единственным обязательным элементом синтаксического обозначения атрибута. Тип атрибута в нотации UML иногда определяется в зависимости от языка программирования, который предполагается использовать для реализации данной модели. В простейшем случае тип атрибута указывается строкой текста, имеющей осмысленное значение в пределах пакета или модели, к которым относится рассматриваемый класс. Исходное значение служит для задания некоторого начального

значения для соответствующего атрибута в момент создания отдельного экземпляра класса.

Операция (operation) представляет собой некоторый сервис, который предоставляет каждый экземпляр класса по определенному требованию. Совокупность операций характеризует функциональный аспект поведения класса. Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения и, возможно, строки-свойства данной операции:

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

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

Квантор видимости для операции может быть опущен. Имя операции представляет собой строку текста, которая используется в качестве идентификатора соответствующей операции и поэтому должна быть уникальной в пределах данного класса. Имя операции – единственный обязательный элемент синтаксического обозначения операции. Список параметров выступает перечнем формальных параметров, разделенных запятой. Выражение типа возвращаемого значения также является зависимой от языка реализации спецификацией типа или типов значений параметров, которые возвращаются объектом после выполнения соответствующей операции. Двоеточие и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения.

Имена операций, так же как и атрибутов, записываются со строчной (малой) буквы, а их типы - с заглавной (большой) буквы. При этом обязательной частью строки записи операции является наличие имени операции и круглых скобок.

В качестве примеров записи операций можно привести следующие обозначения отдельных операций:

- +создать()
- +нарисовать(форма: Многоугольник = прямоугольник, цвет_заливки: Color = (0, 0, 255))

- запросить_счет_клиента(номер_счета:Integer):Currency
- выдать_сообщение():{"Ошибка деления на ноль"}

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

- отношение зависимости, или dependency relationship (рис. 9);
- отношение ассоциации, или association relationship (рис. 10 – 14);
- отношение обобщения, или generalization relationship (рис. 15);
- отношение реализации, или realization relationship.

Каждое из этих отношений имеет собственное графическое представление на диаграмме, которое отражает взаимосвязи между объектами соответствующих классов.

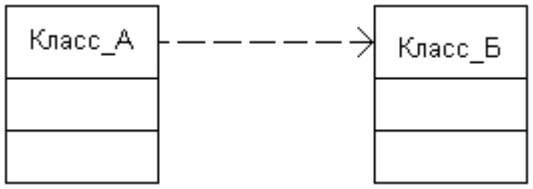

Рис. 9. Графическое изображение отношения зависимости на диаграмме классов

Рис. 10. Графическое изображение отношения бинарной ассоциации между классами

Рис. 11. Графическое изображение тернарной ассоциации между тремя классами

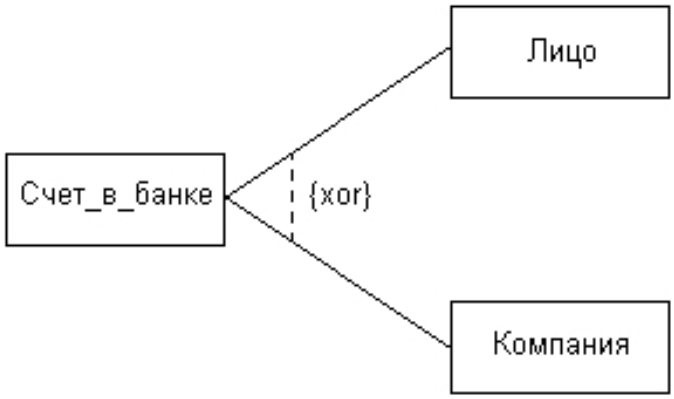

Рис. 12. Графическое изображение исключающей ассоциации между тремя классами

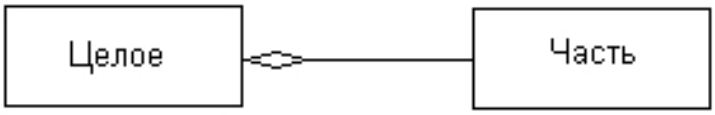

Рис. 13. Графическое изображение отношения агрегации в языке UML

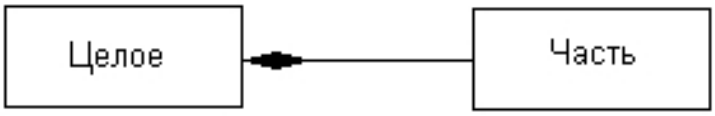

Рис. 14. Графическое изображение отношения композиции в языке UML

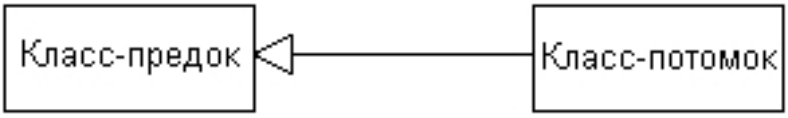

Рис. 15. Графическое изображение отношения обобщения в языке UML

Однако для большинства физических систем, кроме самых простых и тривиальных, статических представлений совершенно недостаточно для моделирования процессов функционирования подобных систем как в целом, так и их отдельных подсистем и элементов.

Диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее – одного экземпляра определенного класса, т.е. моделирует все возможные изменения в состоянии конкретного объекта. При этом изменение состояния объекта может быть вызвано внешними воздействиями со стороны других объектов или извне. Именно для описания реакции объекта на подобные внешние воздействия и используются диаграммы состояний.

Главное предназначение этой диаграммы - описать возможные последовательности состояний и переходов, которые в совокупности характеризуют поведение элемента модели в течение его жизненного цикла. Диаграмма состояний представляет динамическое поведение сущностей на основе спецификации их реакции на восприятие некоторых конкретных событий. Диаграмма состояний, по существу, является графом специального вида, который представляет некоторый автомат.

Формализм обычного автомата основан на выполнении следующих обязательных условий:

1. Автомат не запоминает историю перемещения из состояния в состояние.
2. В каждый момент времени автомат может находиться в одном и только в одном из своих состояний.
3. Хотя процесс изменения состояний автомата происходит во времени, явно концепция времени не входит в формализм автомата.
4. Количество состояний автомата должно быть обязательно конечным (в языке UML рассматриваются только конечные автоматы), и все они должны быть специфицированы явным образом.
5. Граф автомата не должен содержать изолированных состояний и переходов.
6. Автомат не должен содержать конфликтующих переходов.

Таким образом, правила поведения объекта, моделируемого некоторым автоматом, определяются, с одной стороны, общим формализмом автомата, а с другой - его графическим изображением в языке UML в форме конкретной диаграммы состояний.

Рис. 16. Графическое изображение состояний на диаграмме состояний

Понятие *состояния (state)* является фундаментальным не только в метамодели языка UML, но и в прикладном системном анализе. Вся концепция динамической системы основывается на понятии состояния системы. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта (рис. 16).

Имя состояния представляет собой строку текста, которая раскрывает содержательный смысл данного состояния. Имя всегда записывается с за-

главной буквы. Поскольку состояние системы является составной частью процесса ее функционирования, рекомендуется в качестве имени использовать глаголы в настоящем времени (звенит, печатает, ожидает) или соответствующие причастия (занят, свободен, передано, получено).

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния другим. Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. В этом случае говорят, что переход срабатывает, или происходит срабатывание перехода.

Рис. 17. Графическое изображение состояний на диаграмме состоянии: а – начального; б – конечного

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

Термин «*событие» (event)* требует отдельного пояснения, поскольку является самостоятельным элементом языка UML. Формально событие представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

Сторожевое условие (guard condition), если оно есть, всегда записывается в прямых скобках после события и представляет собой некоторое булевское выражение. Из контекста диаграммы состояний должна явно следовать семантика этого выражения, а для его записи может использоваться синтаксис языка объектных ограничений.

Введение для перехода сторожевого условия позволяет явно специфицировать семантику его срабатывания. Если сторожевое условие принимает значение "истина", то соответствующий переход может сработать, в результате чего объект перейдет в целевое состояние (рис. 18).

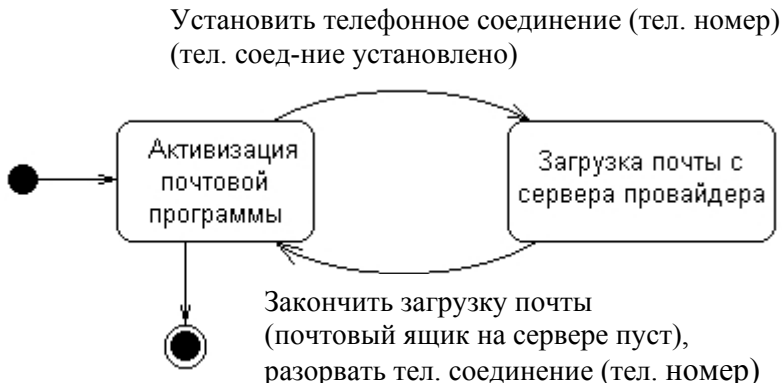

Рис. 18. Диаграмма состояний для моделирования почтовой программы-клиента

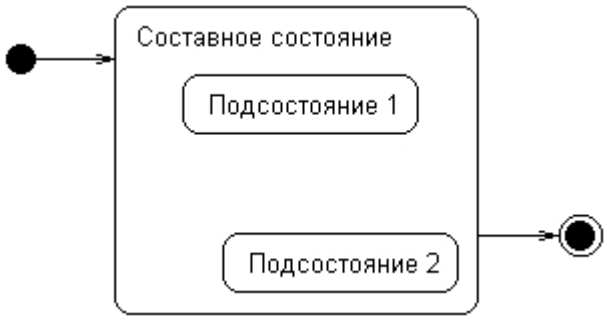

Рис 19. Графическое представление составного состояния с двумя вложенными в него последовательными подсостояниями

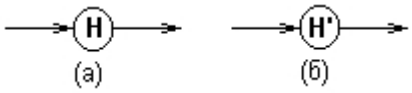

Рис. 20. Графическое изображение: а – недавнего; б – давнего исторического состояния

Составное состояние (composite state) - сложное состояние из других вложенных в него состояний. Последние будут выступать по отношению к первому как *подсостояния (substate)*. Хотя между ними существует отношение композиции, графически все вершины диаграммы, которые соответствуют вложенным состояниям, изображаются внутри символа составного состояния (рис. 19). В этом случае его размеры увеличиваются, чтобы вместить в себя все подсостояния.

Историческое состояние (history state) применяется в контексте составного состояния (рис. 20). Оно используется для запоминания того из последовательных подсостояний, которое было текущим в момент выхода из составного состояния.

Задание

На основании ранее построенной концептуальной модели выявить основные элементы (классы) программной системы, их структуру и отношения. Построить диаграмму классов для проектируемой системы. Уточнить модель за счет уточнения атрибутов и методов классов.

На основании концептуальной модели и статического представления логической модели проанализировать изменение состояний классов про-

граммной системы и построить диаграммы состояний классов, имеющих несколько состояний.

Порядок выполнения работы

1. В Microsoft Office Visio открыть файл, созданный ранее.

2. На нижней панели страниц добавить еще одну страницу в файл, переименовать страницу («Диаграмма классов»).

3. Используя только фигуры с закладки UML Static Structure, построить диаграмму классов.

4. Добавить еще необходимое количество страниц в файл с соответствующими именами.

5. Используя только фигуры с закладки UML Statechart, построить диаграммы состояний.

ПРОЕКТИРОВАНИЕ ПС. ПОСТРОЕНИЕ ДИНАМИЧЕСКОГО ПРЕДСТАВЛЕНИЯ ЛОГИЧЕСКОЙ МОДЕЛИ ПРОГРАММНОЙ СИСТЕМЫ

Общие сведения

Для моделирования взаимодействия объектов в языке UML используются соответствующие диаграммы взаимодействия. Говоря об этих диаграммах, имеют в виду два аспекта взаимодействия. Во-первых, взаимодействия объектов можно рассматривать во времени, и тогда для представления временных особенностей передачи и приема сообщений между объектами используется диаграмма последовательности. Временной аспект поведения может иметь существенное значение при моделировании синхронных процессов, описывающих взаимодействия объектов. Именно для этой цели в языке UML используются диаграммы последовательности.

Во-вторых, можно рассматривать структурные особенности взаимодействия объектов. Для представления структурных особенностей передачи и приема сообщений между объектами используется диаграмма кооперации.

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени (рис. 21).

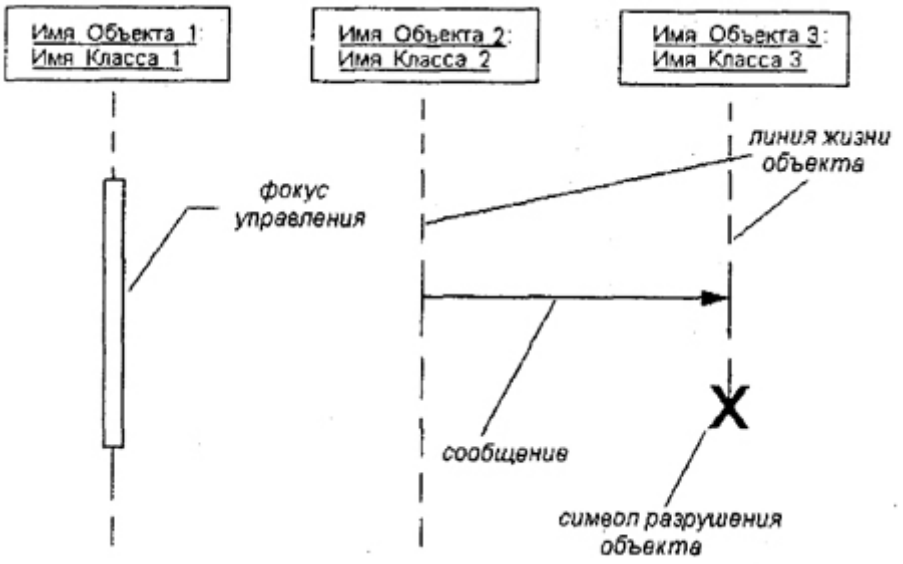

Рис. 21. Различные графические примитивы диаграммы последовательности

Крайним слева на диаграмме изображается объект, который выступает инициатором взаимодействия (объект 1 на рис. 21), правее – другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый степенью активности этих объектов при взаимодействии друг с другом (рис. 22 – 23).

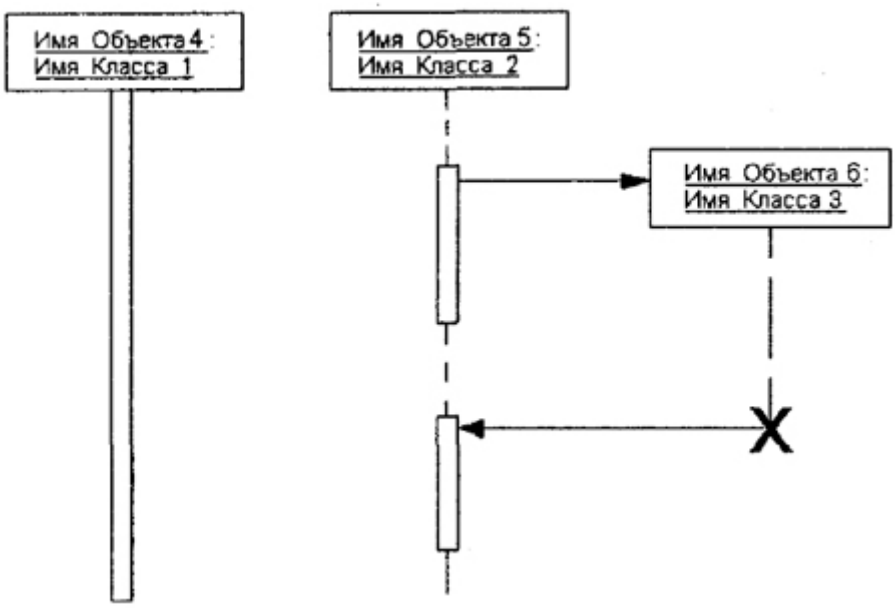

Рис. 22. Графическое изображение различных вариантов линий жизни и фокусов управления объектов

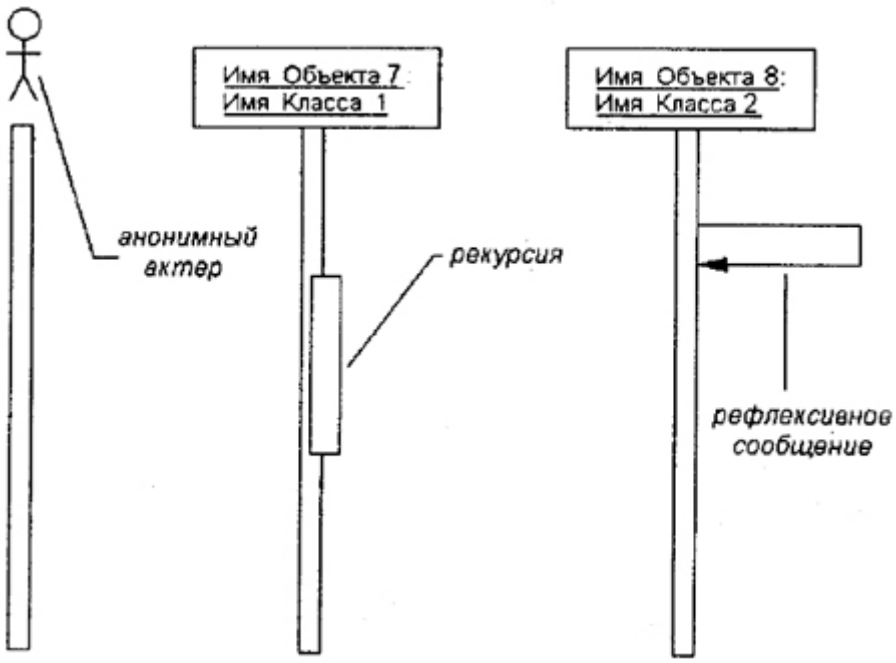

Рис. 23. Графическое изображение актера, рекурсии и рефлексивного сообщения на диаграмме последовательности

Каждое взаимодействие описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой. В этом смысле *сообщение (message)* представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи, тем объектом, которому это сообщение отправлено (рис. 24).

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

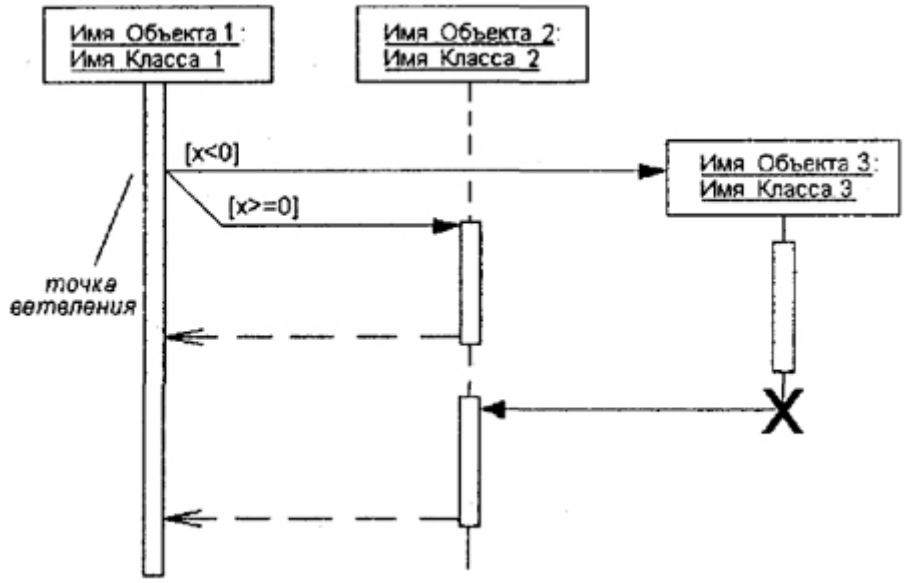

Рис. 24. Графическое изображение различных видов сообщений между объектами на диаграмме последовательности

В языке UML могут встречаться несколько разновидностей сообщений, каждое из которых имеет свое графическое изображение (рис. 25):

- первая разновидность сообщения (рис. 25, *а*) является наиболее распространенной и используется для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления;
- вторая разновидность сообщения (рис. 25, *б*) используется для обозначения простого (не вложенного) потока управления;
- третья разновидность (рис. 25, *в*) явно обозначает асинхронное сообщение между двумя объектами в некоторой процедурной последовательности. Примером такого сообщения может служить прерывание операции при возникновении исключительной ситуации;
- наконец, последняя разновидность сообщения (рис. 25, *г*) используется для возврата из вызова процедуры. Примером может служить простое сообщение о завершении некоторых вычислений без предоставления результата расчетов объекту-клиенту.

Рис. 25. Графическое изображение различных типов сообщений на диаграмме кооперации: а – операция, или вложенный поток управления; б – невложенный поток управления; в – асинхронное сообщение; г – вызов процедуры

Сообщения могут иметь собственное обозначение операции, вызов которой они инициируют у принимающего объекта. В этом случае рядом со стрелкой записывается имя операции с круглыми скобками, в которых могут указываться параметры или аргументы соответствующей операции. Если параметры отсутствуют, то скобки все равно должны присутствовать после имени операции. Примерами таких операций могут служить следующие: "выдать клиенту наличными сумму (n)", "установить соединение между абонентами (a, b)", "сделать вводимый текст невидимым ()", "подать звуковой сигнал тревоги ()" (рис. 26).

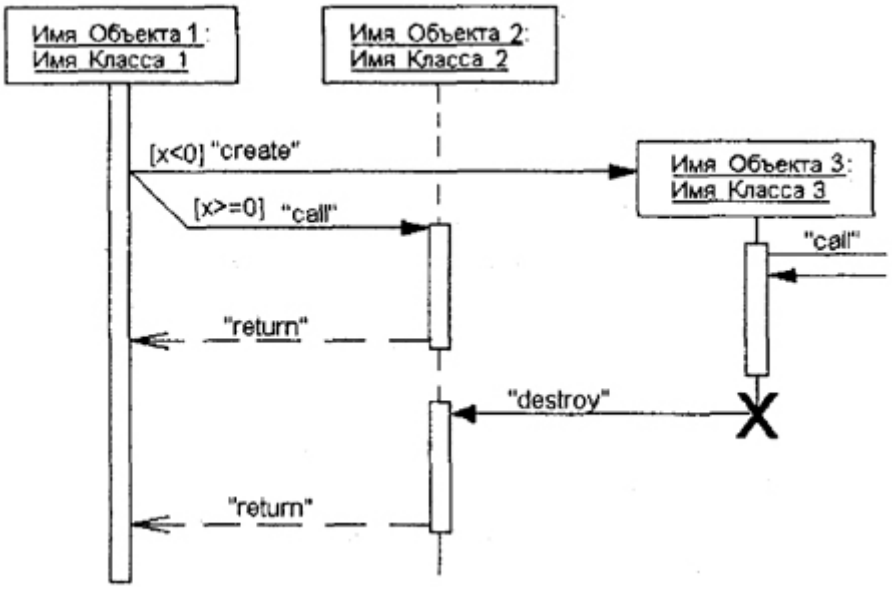

Рис. 26. Диаграмма последовательности со стереотипными значениями сообщений

Задание

На основании концептуальной модели и статического представления логической модели выявить структуру и последовательность взаимодействия классов в рамках конкретных вариантов использования. Построить диаграммы последовательности для всех вариантов использования проектируемой системы.

Порядок выполнения работы

1. В Microsoft Office Visio открыть файл проекта.

2. Добавить необходимое количество страниц в файл с соответствующими именами вида “Диаграмма последовательности для <_вариант-использования_>”.

3. Используя только фигуры с закладки UML Sequence, построить диаграммы последовательности для всех вариантов использования.

ПРОЕКТИРОВАНИЕ ПС. ПОСТРОЕНИЕ ФИЗИЧЕСКОЙ МОДЕЛИ ПРОГРАММНОЙ СИСТЕМЫ

Общие сведения

Различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения. Для создания конкретной физической системы необходимо некоторым образом реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно физическое представление модели.

Для того чтобы пояснить отличие логического и физического представлений, рассмотрим в общих чертах процесс разработки некоторой программной системы. Ее исходным логическим представлением могут служить структурные схемы алгоритмов и процедур, описания интерфейсов и концептуальные схемы баз данных. Однако для реализации этой системы необходимо разработать исходный текст программы на некотором языке программирования (C++, Pascal, Basic/VBA, Java). При этом уже в тексте программы планируется такая организация программного кода, которая предполагает его разбиение на отдельные модули.

Тем не менее исходные тексты программы еще не являются окончательной реализацией проекта, хотя и служат фрагментом его физического представления. Очевидно, программная система может считаться реализованной в том случае, когда она будет способна выполнять функции своего целевого предназначения. А это возможно, только если программный код системы будет реализован в форме исполняемых модулей, библиотек классов и процедур, стандартных графических интерфейсов, файлов баз данных. Именно эти компоненты являются необходимыми элементами физического представления системы.

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диа-

грамма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Основными графическими элементами диаграммы компонентов выступают компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

- визуализации общей структуры исходного кода программной системы;
- спецификации исполнимого варианта программной системы;
- обеспечения многократного использования отдельных фрагментов программного кода;
- представления концептуальной и физической схем баз данных.

Для представления физических сущностей в языке UML применяется специальный термин - *компонент (component)*. Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели.

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и некоторых знаков препинания (рис. 27).

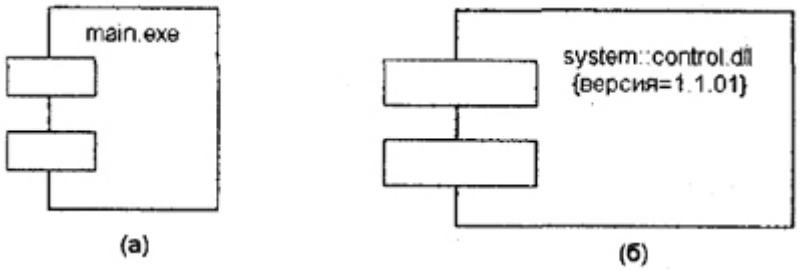

Рис. 27. Графическое изображение компонента в языке UML: а – простое; б – расширенное

В качестве простых имен принято использовать имена исполняемых файлов (с указанием расширения exe после точки-разделителя), динамических библиотек (расширение dll), Web-страниц (расширение html), имена текстовых файлов (расширения txt или doc) или файлов справки (hip), имена файлов баз данных (DB) или файлов с исходными текстами программ (расширения h, cpp для языка C++, расширение Java для языка Java), скрипты (pi, asp) и др.

В языке UML выделяют три вида компонентов:

- компоненты развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки с расширением dll, Web-страницы на языке разметки гипертекста с расширением html и файлы справки с расширением hlp;
- компоненты-рабочие продукты. Как правило, это файлы с исходными текстами программ, например с расширениями h или cpp для языка C++;
- компоненты исполнения, представляющие исполнимые модули - файлы с расширением exe;

Отношение зависимости на диаграмме компонентов изображается пунктирной линией со стрелкой, направленной от клиента (зависимого элемента) к источнику (независимому элементу).

Зависимости могут отражать связи модулей программы на этапе компиляции и генерации объектного кода. В другом случае зависимость может отражать наличие в независимом компоненте описаний классов, которые используются в зависимом компоненте для создания соответствующих объектов.

Так, например, изображенный ниже фрагмент диаграммы компонентов (рис. 28) представляет информацию о том, что компонент с именем "main.exe" зависит от импортируемого интерфейса IDialog, который, в свою очередь, реализуется компонентом с именем "image.java". Для второго компонента этот же интерфейс является экспортируемым.

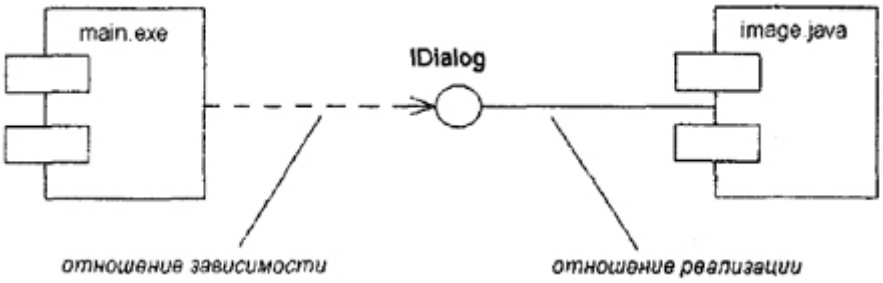

Рис. 28. Фрагмент диаграммы компонентов с отношением зависимости

На диаграмме компонентов могут быть представлены отношения зависимости между компонентами и реализованными в них классами. Эта информация имеет важное значение для обеспечения согласования логического и физического представлений модели системы. Разумеется, измене-

ния в структуре описаний классов могут привести к изменению компонента. Ниже приводится фрагмент зависимости подобного рода, когда некоторый компонент зависит от соответствующих классов (рис. 29).

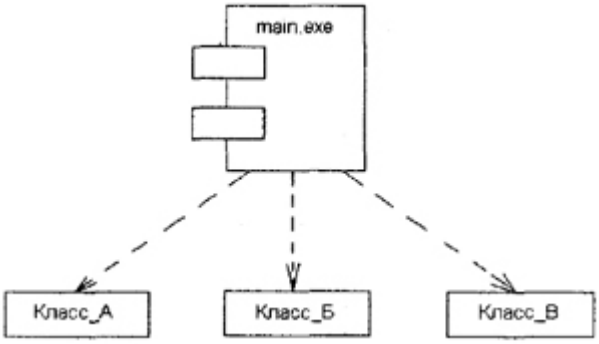

Рис. 29. Графическое изображение зависимости между компонентом и классам

Разработка диаграммы компонентов предполагает использование информации как о логическом представлении модели системы, так и об особенностях ее физической реализации. До начала разработки необходимо принять решения о выборе вычислительных платформ и операционных систем, на которых предполагается реализовывать систему, а также о выборе конкретных баз данных и языков программирования.

После этого можно приступать к общей структуризации диаграммы компонентов. В первую очередь необходимо решить, из каких физических частей (файлов) будет состоять программная система.

После общей структуризации физического представления системы необходимо дополнить модель интерфейсами и схемами базы данных. Включение в модель схемы базы данных предполагает спецификацию отдельных таблиц и установление информационных связей между таблицами.

Наконец, завершающий этап построения диаграммы компонентов связан с установлением и нанесением на диаграмму взаимосвязей между компонентами, а также отношений реализации. Они 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.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются 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 едина для системы в целом, поскольку должна всецело отражать особенности ее реализации. Цели, преследуемые при разработке диаграммы развертывания:

- определить распределение компонентов системы по ее физическим узлам;
- показать физические связи между всеми узлами реализации системы на этапе ее исполнения;
- выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Узел (node) представляет собой некоторый физически существующий элемент системы, обладающий некоторым вычислительным ресурсом. В качестве вычислительного ресурса узла может рассматриваться наличие, по меньшей мере, некоторого объема электронной или магнитооптической памяти и/или процессора (рис. 30). В последней версии языка UML понятие узла расширено и может включать в себя не только вычислительные устройства (процессоры), но и другие механические или электронные устройства, такие как датчики, принтеры, модемы, цифровые камеры, сканеры и манипуляторы.

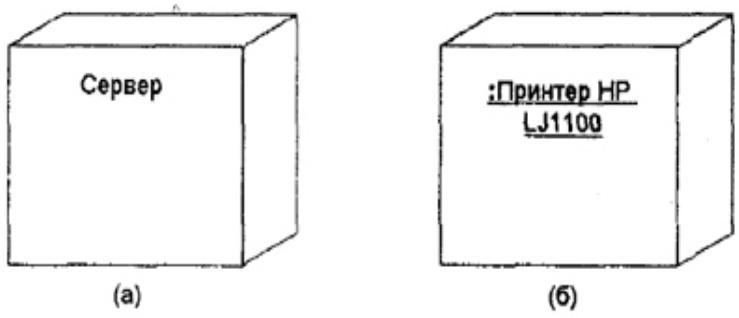

Рис. 30. Графическое изображение узла на диаграмме развертывания: а – узел; б – экземпляр узла

Кроме собственно изображений узлов, на диаграмме развертывания указываются отношения между ними. В качестве отношений выступают физические соединения между узлами и зависимости между ними и компонентами, изображения которых тоже могут присутствовать на диаграммах развертывания.

Соединения являются разновидностью ассоциации и изображаются отрезками линий без стрелок (рис. 31). Наличие такой линии указывает на необходимость организации физического канала для обмена информацией

между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, помеченным значением или ограничением.

Рис. 31. Фрагмент диаграммы развертывания с соединениями между узлами

Кроме соединений, на диаграмме развертывания могут присутствовать отношения зависимости между узлом и развернутыми на нем компонентами. Подобный способ является альтернативой вложенному изображению компонентов внутри символа узла, что не всегда удобно, поскольку делает этот символ излишне объемным. И поэтому при большом количестве развернутых на узле компонентов соответствующую информацию можно представить в форме отношения зависимости (рис. 32).

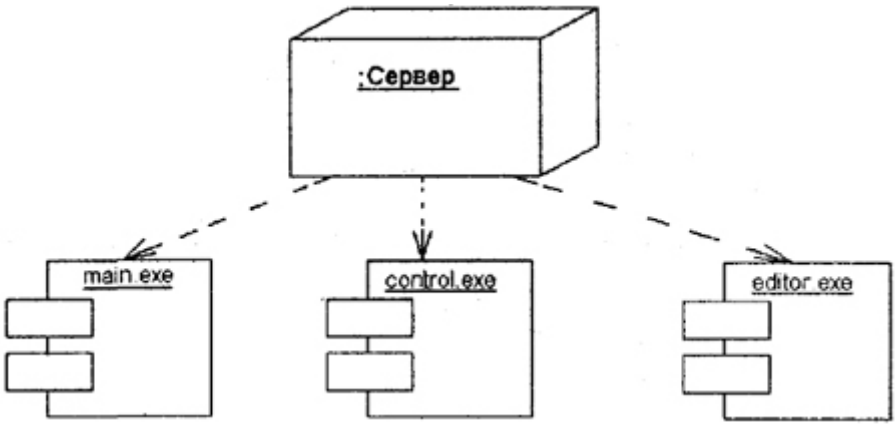

Рис. 32. Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами

Диаграммы развертывания могут иметь более сложную структуру, включающую вложенные компоненты, интерфейсы и другие аппаратные

устройства. На рис. 32 изображен фрагмент физического представления системы удаленного обслуживания клиентов банка. Узлами этой системы являются удаленный терминал (узел-тип) и сервер банка (узел-экземпляр).

Разработка диаграммы развертывания начинается с идентификации всех аппаратных, механических и других типов устройств, которые необходимы для выполнения системой всех своих функций. В первую очередь специфицируются вычислительные узлы системы, обладающие памятью и/или процессором.

Дальнейшее построение диаграммы развертывания связано с размещением всех исполняемых компонентов диаграммы по узлам системы. Если отдельные исполняемые компоненты оказались не размещенными, то подобная ситуация должна быть исключена введением в модель дополнительных узлов, содержащих процессор и память.

Задание

На основании логической модели выявить структуру размещения объектов системы по компонентам и узлам. Построить диаграммы компонентов и развертывания проектируемой системы с указанием размещения классов по компонентам и компонентов по узлам.

Порядок выполнения работы

1. В Microsoft Office Visio открыть файл проекта. Добавить страницу в файл с именем «Диаграмма компонентов».

2. Используя только фигуры с закладки UML Component, построить диаграмму компонентов.

3. Добавить страницу в файл с именем «Диаграмма развертывания».

4. Используя только фигуры с закладки UML Deployment, построить диаграмму развертывания.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. *Вендров, А. М.* Проектирование программного обеспечения экономических информационных систем / А. М. Вендров. – М. : ФиС, 2005. – 352 с. – ISBN 5-279-02144-X.

2. *Буч, Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Гради Буч. – М. : Бином, 2001. – 560 с. – ISBN 5-7989-0067-3.

3. ISTQB Ceritified Software Tester Syllabus. Foundation Level. ISTQB. – [Режим доступа]. – http://www.RSTQB.org.

4. *Леоненков, А.* Самоучитель UML 2 / А. Леоненков. – СПб. : БХВ-Петербург, 2007. – 576 с. – ISBN 978-5-94157-878-8.

5. *Фаулер, М.* UML. Основы. Краткое руководство по унифицированному языку моделирования / М. Фаулер, К. Скотт. – 2-е изд. – М. : Символ-Плюс, 2002. – 192 с. – ISBN 5-93286-032-4.

ОГЛАВЛЕНИЕ

ЯЗЫК ВИЗУАЛЬНОГО МОДЕЛИРОВАНИЯ UML

Методические указания к курсовой работе
по дисциплине «Разработка и стандартизация программных средств и технологий»

Составители
КОНУШИН Андрей Владимирович
МАЗАНОВА Валентина Ивановна

Ответственный за выпуск – зав. кафедрой профессор Д. В. Александров

Подписано в печать 25.05.12.
Формат 60×84/16. Усл. печ. л. 1,86. Тираж 75 экз.
Заказ
Издательство
Владимирского государственного университета
имени Александра Григорьевича и Николая Григорьевича Столетовых.
600000, Владимир, ул. Горького, 87.