Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования
«Севастопольский государственный университет»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к лабораторным работам по дисциплине

"Технологии создания программных продуктов"

Часть 1

для студентов, обучающихся по направлению

09.03.02 “Информационные системы и технологии”

очной и заочной форм обучения

Севастополь
2015

УДК 004.732

Методические указания к лабораторным занятиям по дисциплине "Технологии создания программных продуктов". Часть 1 / Сост. Строганов В.А., Дрозин А.Ю. — Севастополь: Изд-во СевГУ, 2015— 48 с.

Методические указания предназначены для проведения лабораторных работ по дисциплине “Технологии создания программных продуктов“. Целью методических указаний является помощь студентом в выполнении лабораторных работ. Излагаются теоретические и практические сведения необходимые для выполнения лабораторной работы, программы работы, требования к содержанию отчета. Целью лабораторного практикума является приобретение студентами практических навыков составления диаграмм на языке UML с использованием соответствующих CASE-средств, применения диаграмм при разработке программных систем.

Методические указания рассмотрены и утверждены на методическом семинаре и заседании кафедры информационных систем

(протокол № 1 от 31 августа 2015 г.)

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: Кротов К.В., канд. техн. наук, доцент кафедры ИС

Содержание

Общие положения

1. Цель и задачи лабораторных работ

Цель настоящих лабораторных работ состоит в исследовании основных принципов построения диаграмм языка UML при проектировании программных продуктов.

Задачами выполнения лабораторных работ являются:

– углубленное изучение основных теоретических положений дисциплины «Технологии компьютерного проектирования»;

– получение практических навыков разработки программных систем, их разработки с помощью унифицированного языка моделирования UML, используя современные CASE-средства.

2. Описание лабораторной установки

Объектом исследования в лабораторных работах являются способы описания проектируемой системы с использованием унифицированного языка моделирования UML. В качестве лабораторной установки используется персональный компьютер и программное обеспечение — UML-редактор с возможностью синтеза программного кода на основе UML-диаграмм. Рекомендуется использовать StarUML, ArgoUML. Для компиляции и отладки программного кода рекомендуется использовать среду разработки Microsoft Visual Studio.

3. Порядок выполнения лабораторных работ

Варианты заданий студенты получают у преподавателя. На выполнение лабораторной работы №1 отводится 2 часа, на выполнение лабораторных работы №2 — 3 часа. Лабораторные работы №3, №4 и №5 выполняются за 4 часа каждая.

Объем работы распределяется следующим образом: в ходе домашней самостоятельной подготовки студенты изучают необходимый теоретический материал. В ходе аудиторных занятий студенты выполняют построение UML-диаграмм, а для лабораторной работы №2 также генерацию кода программы, компиляцию и проверку работоспособности программ.

При выполнении лабораторных работ №№1 — 5 необходимо:

- провести анализ предметной области;
- определить объекты и отношения между объектами данной предметной области;
- разработать диаграмму, указанную в варианте задания;
- построить соответствующую диаграмму, используя систему StarUML;
- провести проверку правильного составления диаграммы;
- проанализировать результаты работы, сделать выводы.

4. Содержание отчета

Отчеты по лабораторной работе оформляются каждым студентом индивидуально. Отчет должен включать:

- название и номер лабораторной работы;
- цель работы; постановку задачи;
- описание предметной области;
- разработанную диаграмму и пояснения к ней (подробное описание элементов диаграммы и связей между ними);
- выводы по результатам работы.

5.1. Порядок работы в UML-редакторе StarUML

StarUML представляет собой UML-редактор с возможностью синтеза программного кода на основе UML-диаграмм. Ниже рассматривается общий порядок работы в StarUML.

5.1.1. Создание нового проекта

Для создания нового проекта следует воспользоваться диалогом создания проекта (пункт меню File->New Project by Approach либо сочетание клавиш Ctrl + I). В диалоговом окне (рисунок 1) рекомендуется выбрать вариант «Empty project» (пустой проект).

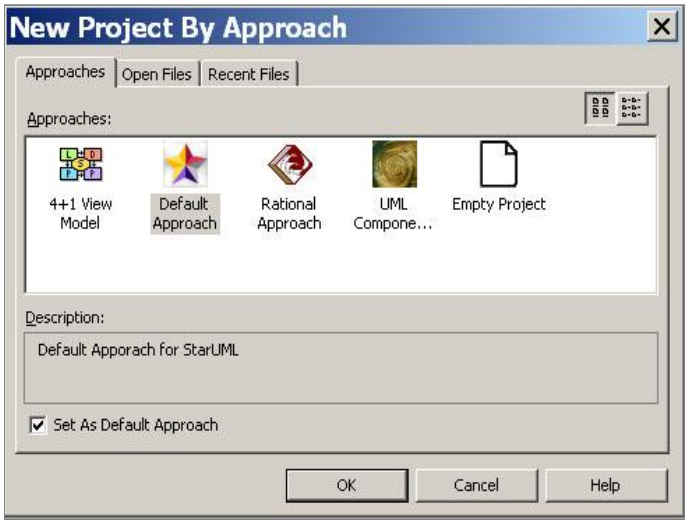

Рисунок 1 – Создание нового проекта

В результате будет создан новый проект, не содержащий никаких UML-диаграмм. Вид главного окна StarUML после создание нового проекта показан на рисунке 2.

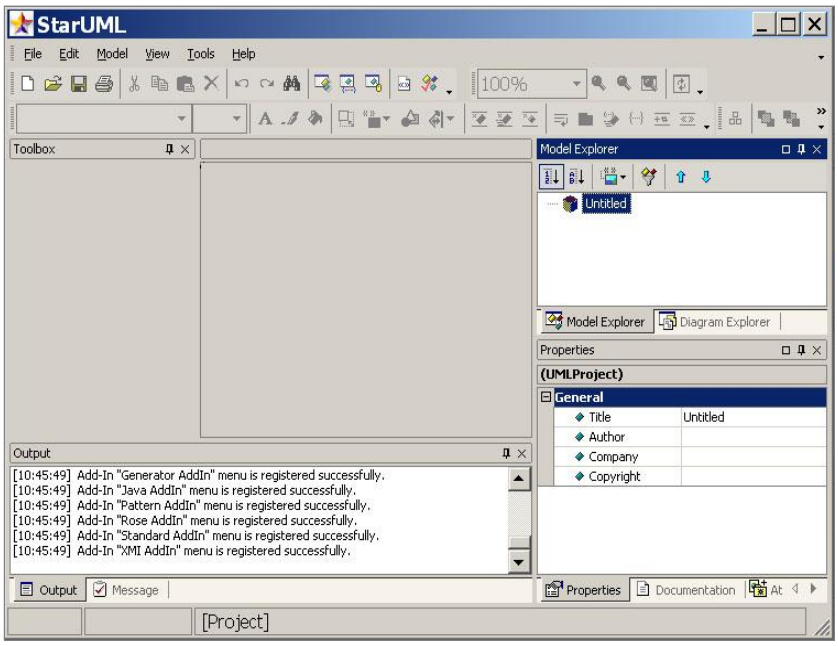

Рисунок 2 – Главное окно StarUML

В центральной части окна располагается основное поле (в данный момент – пустое), в котором будут отображаться UML-диаграммы.

В левой части окна располагается панель «Toolbox», которая будет содержать инструменты для создания элементов диаграмм (классы, интерфейсы, связи и т. п.). Содержимое панели «Toolbox» определяется типом редактируемой в данный момент UML-диаграммы.

В правой части главного окна располагаются панели «Model Explorer» (Обозреватель моделей) и «Properties» (Свойства). Панель «Model Explorer» содержит древовидную структуру текущего проекта, включающую модели, подсистемы, пакеты и входящие в их состав диаграммы различных типов (диаграммы классов, прецедентов и проч.).

5.1.2. Создание модели

Для добавления новой модели к текущему проекту необходимо воспользоваться контекстным меню панели «Model Explorer», как показано на рисунке 3.

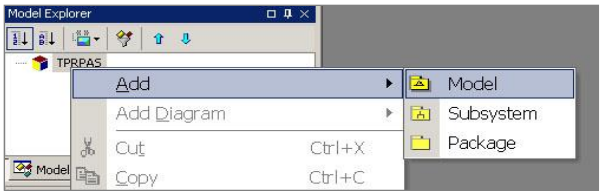

Рисунок 3 – Создание новой модели

В результате новая модель будет создана и добавлена в дерево проекта (рисунок 4).

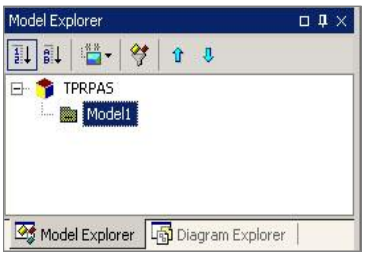

Рисунок 4 – Дерево проекта

5.1.3. Создание и редактирование диаграммы классов

После того, как создана модель, можно приступать к созданию диаграмм классов. Для добавления диаграммы классов следует воспользоваться контекстным меню для модели в панели «Model Explorer», как показано на рисунке 5.

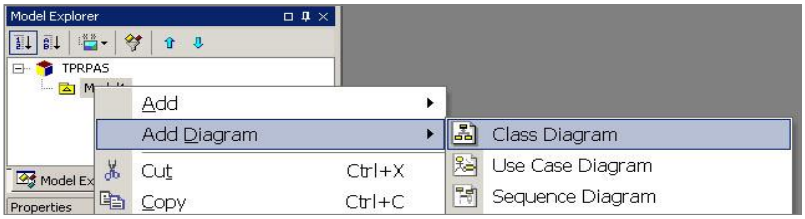

Рисунок 5 – Создание диаграммы классов

В результате вновь созданная диаграмма классов будет открыта в центральной части окна StarUML (рисунок 6).

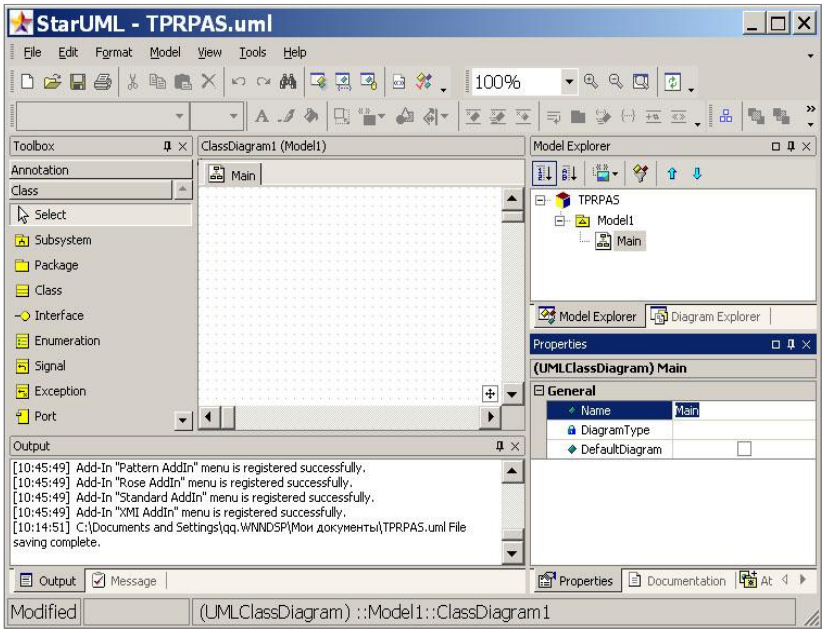

Рисунок 6 – Редактирование диаграммы классов

В панели инструментов («Toolbox») при этом появятся инструменты для создания элементов диаграммы классов (рисунок 7).

Для создания нового класса необходимо выбрать элемент «Class» на панели инструментов и нажать мышкой в центральном окне, в котором отображается диаграмма классов. В результате будет создан класс (рисунок 8). Для изменения имени класса можно воспользоваться панелью «Properties».

Рисунок 7 – Панель инструментов

Для добавления атрибутов и методов класса используется контекстное меню («Add / Attribute» и «Add / Operation» соответственно). Имена методов и атрибутов могут быть изменены с помощью панели «Properties».

Интерфейсы создаются аналогично классам.

Для добавления связей между классами и интерфейсами следует выбрать связь нужного типа (Association, Composition, Generalization) на панели инструментов «Toolbox», соединить нужные классы и при необходимости задать параметры связи на панели «Properties».

В качестве примера на рисунке 8 показана диаграмма классов, включающая интерфейс `CanGo` (может передвигаться) с методом `go()` и два класса (`Car` и `Driver`), реализующие этот интерфейс. Между классами `Car` и `Driver` установлено отношение ассоциации `drives`.

Для редактирования и удаления методов и атрибутов класса можно воспользоваться панелью «Properties» для класса (раздел General / Operations) или двойным нажатием левой кнопки мышки на нужном атрибуте (или методе) вызвать меню редактирования атрибуту (метода), как показано на рисунке 8.

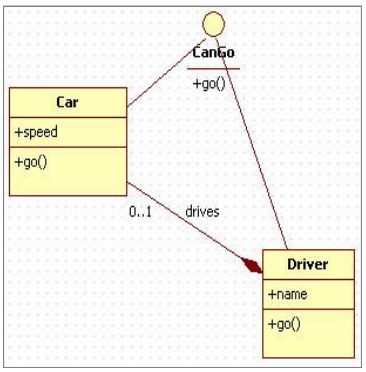

Рисунок 8 – Редактирование метода класса

5.1.4. Генерация кода программы

StarUML позволяет генерировать программный код на основе диаграммы классов. Это дает возможность выполнить проектирование системы с помощью визуального редактора диаграмм, после чего автоматически получить шаблон текста программы, содержащий объявление классов, интерфейсов, атрибутов и методов. В результате программист получает исходные тексты классов и реализует необходимую функциональность согласно спецификации методов, не отвлекаясь на общую структуру проекта.

Для генерации программного кода следует воспользоваться пунктом главного меню Tools / C# / Generate Code. Далее следуют диалоговые окна, в которых нужно выбрать модель и классы, исходный код которых будет генерироваться, папку, в которой он будет сохранен, способ сохранения (все классы в одном файле или отдельный файл для каждого класса), задать формат заголовка файла. Для всех этих параметров можно оставить значения по умолчанию.

Для диаграммы классов, показанной на рисунке 8, будет сгенерирован такой программный код:

```
//  Generated by StarUML(tm) C# Add-In
//  @ Project : TPRPAS
//  @ File Name : Model1.cs
//  @ Date : 27.03.2012
//  @ Author :
public class Car : CanGo{
     public object speed ;
     public void go(){
     }
}
public class Driver : CanGo{
     public object name ;
     public void go(){
     }
}
public interface CanGo {
     void go();
}
```

Лабораторная работа №1

«Исследование способов построения диаграмм прецедентов»

1. Цель работы

Исследование правил построения диаграмм прецедентов на этапе анализа предметной области. Исследование отношений на диаграмме прецедентов.

2. Основные положения

2.1. Прецеденты

Конструкция или стандартный элемент языка UML ***прецедент*** (англ. use case – вариант использования) применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый прецедент определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. Диаграмма вариантов использования может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст получил название примечания или сценария.

Отдельный прецедент обозначается на диаграмме эллипсом, внутри которого содержится его краткое название или имя в форме глагола с пояснительными словами (рисунок 1).

Рисунок 1 – Графическое обозначение прецедента

Цель прецедента заключается в том, чтобы определить законченный аспект или фрагмент поведения некоторой сущности без раскрытия внутренней структуры этой сущности. В качестве такой сущности может выступать исходная система или любой другой элемент модели, который обладает собственным поведением, подобно подсистеме или классу в модели системы.

Каждый прецедент соответствует отдельному сервису, который моделируемая сущность или система предоставляет 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.

Прецеденты описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности. Прецеденты могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы. Множество прецедентов в целом должно определять все возможные стороны ожидаемого поведения системы. Для удобства множество прецедентов можно рассматриваться как отдельный пакет.

С системно-аналитической точки зрения прецеденты могут применяться как для спецификации внешних требований к проектируемой системе, так и для спецификации функционального поведения уже существующей системы. Кроме этого, прецеденты неявно устанавливают требования, определяющие, как пользователи должны взаимодействовать с системой, чтобы иметь возможность корректно работать с предоставляемыми данной системой сервисами!

Применение прецедентов на всех уровнях диаграммы позволяет не только достичь требуемого уровня унификации обозначений для представления функциональности подсистем и системы в целом, но и является мощным средством последовательного уточнения требований к проектируемой системе на основе полууровневого спуска от пакетов системы к операциям классов. С другой стороны, модификация отдельных операций класса может оказать обратное влияние на уточнение сервиса соответствующего прецедента, т. е. реализовать эффект обратной связи с целью уточнения спецификаций или требований на уровне пакетов системы.

2.2. Актеры

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. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного прецедента. Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается конкретное имя актера (рисунок 2).

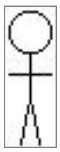

Рисунок 2 - Графическое обозначение актера

В некоторых случаях актер может обозначаться в виде прямоугольника класса с ключевым словом "Actor" и обычными составляющими элементами класса. Имена актеров должны записываться заглавными буквами и следовать рекомендациям использования имен для типов и классов модели. При этом символ отдельного актера связывает соответствующее описание актера с конкретным именем. Имена абстрактных актеров, как и других абстрактных элементов языка UML, рекомендуется обозначать курсивом.

Актеры используются для моделирования внешних по отношению к проектируемой системе сущностей, которые взаимодействуют с системой и используют ее в качестве отдельных пользователей. В качестве актеров могут выступать другие системы, подсистемы проектируемой системы или отдельные классы. Важно понимать, что каждый актер определяет некоторое согласованное множество ролей, в которых могут выступать пользователи данной системы в процессе взаимодействия с ней. В каждый момент времени с системой взаимодействует вполне определенный пользователь, при этом он играет или выступает в одной из таких ролей. Наиболее наглядный пример актера — конкретный пользователь системы со своими собственными параметрами аутентификации.

Любая сущность, которая согласуется с подобным неформальным определением актера, представляет собой экземпляр или пример актера. Для моделируемой системы актерами могут быть как субъекты-пользователи, так и другие системы. Поскольку пользователи системы всегда являются внешними по отношению к этой системе, то они всегда представляются в виде актеров.

Так как в общем случае актер всегда находится вне системы, его внутренняя структура никак не определяется. Для актера имеет значение только его внешнее представление, т. е. то, как он воспринимается со стороны системы. Актеры взаимодействуют с системой посредством передачи и приема сообщений от прецедентов. Сообщение представляет собой запрос актером сервиса от системы и получение этого сервиса. Это взаимодействие может быть выражено посредством ассоциаций между отдельными актерами и прецедентами или классами. Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами.

2.3. Отношения на диаграмме прецедентов

Между компонентами диаграммы прецедентов могут существовать различные отношения, которые описывают взаимодействие экземпляров

одних актеров и прецедентов с экземплярами других актеров и прецедентов. Один актер может взаимодействовать с несколькими прецедентами. В этом случае этот актер обращается к нескольким сервисам данной системы. В свою очередь один прецедент может взаимодействовать с несколькими актерами, предоставляя для всех них свой сервис. Следует заметить, что два прецедента, определенные для одной и той же сущности, не могут взаимодействовать друг с другом, поскольку каждый из них самостоятельно описывает законченный прецедент этой сущности. Более того, прецеденты всегда предусматривают некоторые сигналы или сообщения, когда взаимодействуют с актерами за пределами системы. В то же время могут быть определены другие способы для взаимодействия с элементами внутри системы.

Актер и прецедент связываются отношением ассоциации (association relationship).

Между прецедентами возможны следующие виды отношений:

- отношение расширения (extend relationship): новый прецедент может расширять исходный, называемый также базовым, за счет добавления новых шагов.
- отношение обобщения (generalization relationship): дочерний прецедент наследует последовательность действий от базового и добавляет свою собственную очередность шагов. Дочерний прецедент можно применить там, где применяется родительский.
- отношение включения (include relationship): действия одного прецедента включают действия другого.

3. Варианты заданий и порядок выполнения работы

3.1. Выбрать предметную область из перечисленных в Приложении А.

3.2. Провести анализ выбранной предметной области, выделить ключевых актеров и прецеденты.

3.3. Исследовать отношения между прецедентами.

3.4. По результатам анализа предметной области построить диаграмму прецедентов, содержащую не менее трех актеров и не менее десяти прецедентов. Использовать все типы отношений между прецедентами.

4. Содержание отчета

4.1. Цель работы.

4.2. Задание на работу.

4.3. Словесное описание предметной области.

4.4. Диаграмма прецедентов.

4.5. Словесное описание диаграммы прецедентов с описанием всех актеров, прецедентов и отношений между ними.

4.6. Выводы по результатам работы.

5. Контрольные вопросы

5.1. Что называется прецедентом?
5.2. Для чего используется актер на диаграмме прецедентов?
5.3. Расскажите об отношениях на диаграмме прецедентов.
5.4. Какие аспекты охватывает инженерия программного обеспечения?
5.5. Что такое технологический процесс создания ПО?
5.6. Что представляет собой модель технологического процесса создания ПО?

Лабораторная работа №2

«Исследование способов построения диаграмм классов»

1. Цель работы

Исследование способов описания классов в языке UML, определения атрибутов и операций для класса. Изучение видов связей в диаграмме классов, правил описания и использования интерфейсов.

2.Общие положения

2.1.Классы, атрибуты, операции и обязанности

Классом (Class) в ООП называется описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Графически класс изображается в виде прямоугольника.

Атрибут - это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов или не иметь их вовсе. Атрибут представляет некоторое свойство моделируемой сущности, общее для всех объектов данного класса. Например, у любой стены есть высота, ширина и толщина; при моделировании клиентов можно задавать фамилию, адрес, номер телефона и дату рождения. Таким образом, атрибут является абстракцией данных объекта или его состояния. В каждый момент времени любой атрибут объекта, принадлежащего данному классу, обладает вполне определенным значением. Атрибуты представлены в разделе, который расположен под именем класса; при этом указываются только их имена.

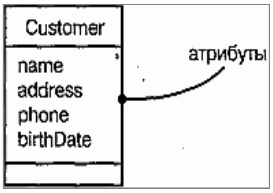

Рисунок 1 – Описание атрибутов класса

При описании атрибута можно явным образом указывать его класс и начальное значение, принимаемое по умолчанию, как продемонстрировано на рисунке 2.

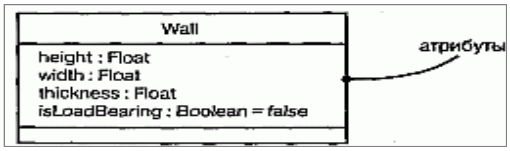

Рисунок 2 – Описание типов и начальных значений атрибутов

Операцией называется реализация услуги, которую можно запросить у любого объекта класса для воздействия на поведение. Иными словами, операция – это абстракция того, что позволено делать с объектом. У всех объектов класса имеется общий набор операций. Класс может содержать любое число операций или не содержать их вовсе. Часто (хотя не всегда) обращение к операции объекта изменяет его состояние или его данные. Операции класса изображаются в разделе, расположенном ниже раздела с атрибутами. При этом можно ограничиться только именами, как показано на рисунке 3.

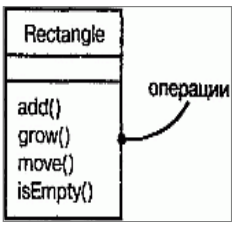

Рисунок 3 – Описание операций класса

Имя операции, как и имя класса, может быть произвольной текстовой строкой. На практике для именования операций используют короткий глагол или глагольный оборот, соответствующий определенному поведению объемлющего класса. Каждое слово в имени операции, кроме самого первого, обычно пишут с заглавной буквы, например *move* или *isEmpty*.

Операцию можно описать более подробно, указав ее сигнатуру, в которую входят имена и типы всех параметров, их значения, принятые по умолчанию, а применительно к функциям – тип возвращаемого значения, как показано на рисунке.

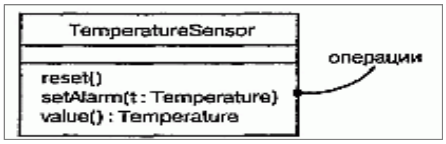

Рисунок 4 – Задание сигнатуры операций

Обязанности (Responsibilities) класса – это своего рода контракт, которому он должен подчиняться. Определяя класс, вы постулируете, что все его объекты имеют однотипное состояние и ведут себя одинаково. Выражаясь абстрактно, соответствующие атрибуты и операции как раз и являются теми свойствами, посредством которых выполняются обязанности класса. Например, класс *Wall* (Стена) отвечает за информацию о высоте, ширине и толщине. Класс *TemperatureSensor* (Датчик Температуры) отвечает за измерение температуры и подачу сигнала тревоги в случае превышения заданного уровня и т.д.

Моделирование классов лучше всего начинать с определения обязанностей сущностей, которые входят в словарь системы. На этом этапе особенно полезны будут такие методики, как применение CRC-карточек и анализ прецедентов. В принципе число обязанностей класса может быть

произвольным, но на практике хорошо структурированный класс имеет по меньшей мере одну обязанность; с другой стороны, их не должно быть и слишком много. При уточнении модели обязанности класса преобразуются в совокупность атрибутов и операций, которые должны наилучшим образом обеспечить их выполнение.

Графически обязанности изображают в особом разделе в нижней части пиктограммы класса (см. рисунок 5). Их можно указать также в примечании.

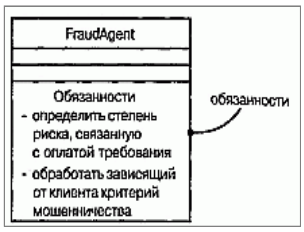

Рисунок 5 – Описание обязанностей класса

2.2.Отношения между классами

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 (dependency relationship);
- отношение ассоциации (association relationship);
- отношение обобщения (generalization relationship).

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

2.2.1.Отношение зависимости

Отношение зависимости в общем случае указывает некоторое семантическое отношение между двумя элементами модели или двумя множествами таких элементов, которое не является отношением ассоциации, обобщения или реализации. Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов ("– –→" или "← – –"). На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от класса-клиента зависимости к независимому классу или классу-источнику. На рисунке 6 изображены два класса: *Класс_А* и *Класс_Б*, при этом *Класс_Б* является источником некоторой зависимости, а *Класс_А* – клиентом этой зависимости.

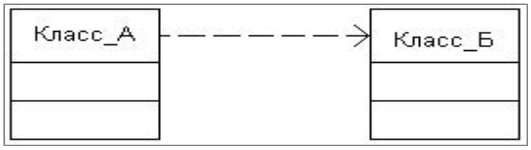

Рисунок 6 – Отношения зависимости на диаграмме классов

Стрелка может помечаться необязательным, но стандартным ключевым словом в кавычках и необязательным индивидуальным именем. Для отношения зависимости предопределены ключевые слова, которые обозначают некоторые специальные виды зависимостей. Эти ключевые слова (стереотипы) записываются в кавычках рядом со стрелкой, которая соответствует данной зависимости. Примеры стереотипов для отношения зависимости представлены ниже:

- "access" – служит для обозначения доступности открытых атрибутов и операций класса-источника для классов-клиентов;
- "bind" – класс-клиент может использовать некоторый шаблон для своей последующей параметризации;
- "derive" – атрибуты класса-клиента могут быть вычислены по атрибутам класса-источника;
- "import" – открытые атрибуты и операции класса-источника становятся частью класса-клиента, как если бы они были объявлены непосредственно в нем;
- "refine" – указывает, что класс-клиент служит уточнением класса-источника в силу причин исторического характера, когда появляется дополнительная информация в ходе работы над проектом.

2.2.2.Отношение ассоциации

Отношение ассоциации соответствует наличию некоторой связи между классами. Данное отношение обозначается сплошной линией с дополнительными специальными символами, которые характеризуют отдельные свойства конкретной ассоциации. В качестве дополнительных специальных символов могут использоваться имя ассоциации, а также имена и кратность классов-ролей ассоциации. Имя ассоциации является необязательным элементом ее обозначения. Если оно задано, то записывается с заглавной (большой) буквы рядом с линией соответствующей ассоциации.

Наиболее простой случай данного отношения – бинарная ассоциация. Она связывает в точности два класса и, как исключение, может связывать класс с самим собой. Для бинарной ассоциации на диаграмме может быть указан порядок следования классов с использованием треугольника в форме стрелки рядом с именем данной ассоциации. Направление этой стрелки указывает на порядок классов, один из которых является первым (со стороны треугольника), а другой — вторым (со стороны вершины треугольника). Отсутствие данной стрелки рядом с именем ассоциации означает, что порядок следования классов в рассматриваемом отношении не определен.

В качестве простого примера отношения бинарной ассоциации рассмотрим отношение между двумя классами – классом "*Компания*" и

классом "*Сотрудник*" (рисунок 7). Они связаны между собой бинарной ассоциацией "*Работа*", имя которой указано на рисунке рядом с линией ассоциации. Для данного отношения определен порядок следования классов, первым из которых является класс "*Сотрудник*", а вторым – класс "*Компания*". Отдельным примером или экземпляром данного отношения может являться пара значений (Петров И.И., "Рога-и-Копыта"). Это означает, что сотрудник Петров И.И. работает в компании "Рога-и-Копыта".

Рисунок 7 – Отношение бинарной ассоциации между классами

Ассоциации более высокой арности в общем случае называются *N*-арной ассоциацией (читается — "эн арная ассоциация"). Такая ассоциация связывает некоторым отношением 3 и более классов, при этом один класс может участвовать в ассоциации более чем один раз. Класс ассоциации имеет определенную роль в соответствующем отношении, что может быть явно указано на диаграмме. Каждый экземпляр *N*-арной ассоциации представляет собой *N*-арный кортеж значений объектов из соответствующих классов. Бинарная ассоциация является частным случаем *N*-арной ассоциации, когда значение *N=2*, и имеет свое собственное обозначение.

N-арная ассоциация графически обозначается ромбом, от которого ведут линии к символам классов данной ассоциации. В этом случае ромб соединяется с символами соответствующих классов сплошными линиями. Обычно линии проводятся от вершин ромба или от середины его сторон. Имя *N*-арной ассоциации записывается рядом с ромбом соответствующей ассоциации.

Порядок классов в *N*-арной ассоциации, в отличие от порядка множеств в отношении, на диаграмме не фиксируется. Некоторый класс может быть присоединен к ромбу пунктирной линией. Это означает, что данный класс обеспечивает поддержку свойств соответствующей *N*-арной ассоциации, а сама *N*-арная ассоциация имеет атрибуты, операции и/или ассоциации. Другими словами, такая ассоциация, в свою очередь, является классом с соответствующим обозначением в виде прямоугольника и является самостоятельным элементом языка UML – ассоциативным классом (Association Class). *N*-арная ассоциация не может содержать символ агрегации ни для какой из своих ролей.

2.2.3.Отношение обобщения

Отношение обобщения является обычным таксономическим отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Данное отношение может использоваться для представления взаимосвязей

между пакетами, классами, вариантами использования и другими элементами языка UML.

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рисунок 8). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие — на более специальный класс (класс-потомок или подкласс).

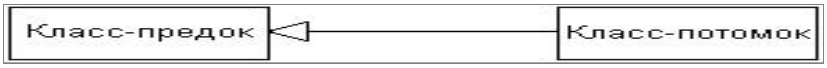

Рисунок 8 – Отношения обобщения на UML-диаграмме классов

3. Варианты заданий и порядок выполнения работы

3.1. Для предметной области, проанализированной в работе № 1, выделить основные классы;

3.2. Для каждого класса определить атрибуты и операции;

3.3. Определить связи между классами;

3.4. Построить диаграмму классов системы, использовать все типы отношений между классами.

4. Содержание отчета

4.1. Цель работы.

4.2. Задание на работу.

4.3. Словесное описание предметной области.

4.4. Диаграмма классов.

4.5. Словесное описание диаграммы классов с описанием каждого класса и отношений между классами.

4.6. Выводы по результатам работы.

5. Контрольные вопросы

5.1. Для чего служит диаграмма классов и как она представляется?

5.2. Каким образом описываются классы в UML.

5.3. Расскажите об атрибутах класса.

5.4. Расскажите об операциях.

5.5. Опишите основные отношения между классами.

5.6. Дайте определение системы и расскажите о ее свойствах.

5.7. Опишите процесс создания системы в соответствии с каскадной моделью, расскажите о каждом из этапов.

Лабораторная работа №3

«Исследование способов построения диаграмм состояний»

1. Цель работы

Исследовать способы описания поведения объекта во времени. Изучить события, действия и условия перехода, а также последовательные и параллельные состояния.

2. Общие положения

2.1. Диаграммы состояний

Рассмотренные в работе №2 диаграммы классов представляют собой логическую модель статического представления моделируемой системы. На таких диаграммах изображаются только взаимосвязи структурного характера, не зависящие от времени или реакции системы на внешние события. 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.

Понимание семантики термина «***состояние***» представляет определенные трудности. Дело в том, что характеристика состояний системы не зависит (или слабо зависит) от логической структуры, зафиксированной в диаграмме классов. Поэтому при рассмотрении состояний системы приходится на время отвлечься от особенностей ее объектной структуры и мыслить совершенно другими категориями, образующими динамический контекст поведения моделируемой системы.

Для моделирования поведения на логическом уровне в языке UML могут использоваться сразу несколько канонических диаграмм: состояний, видов деятельности, последовательности и кооперации, каждая из которых акцентирует внимание на отдельном аспекте функционирования системы. В отличие от других диаграмм, диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее — одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта. 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. 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.

Хотя диаграммы состояний чаще всего используются для описания поведения отдельных экземпляров классов (объектов), но они также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов.

2.2. Состояние

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. Однако семантика состояния в языке UML имеет целый ряд специфических особенностей.

В языке UML под состоянием понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой имеет место выполнение некоторого условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта.

Следует заметить, что не каждый атрибут класса может характеризовать его состояние. Как правило, имеют значение только такие свойства элементов системы, которые отражают динамический или функциональный аспект ее поведения. В этом случае состояние будет характеризоваться некоторым инвариантным условием, включающим в себя только значимые для поведения класса атрибуты и их значения.

Например, инвариант может представлять статическую ситуацию, когда объект находится в состоянии ожидания возникновения некоторого внешнего события. С другой стороны, инвариант используется для моделирования динамических аспектов, когда в ходе процесса выполняются некоторые действия. В этом случае моделируемый элемент переходит в рассматриваемое состояние в момент начала соответствующей деятельности и покидает данное состояние в момент ее завершения.

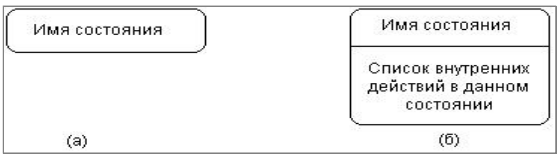

Рисунок 1 – Графическое изображение состояний на диаграмме состояний

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рисунок 1). Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния (рисунок 1, а). В противном случае в первой из них записывается имя состояния, а во второй — список некоторых внутренних действий или переходов в данном состоянии (рисунок 1, 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").

Имя состояния представляет собой строку текста, которая раскрывает содержательный смысл данного состояния. Имя всегда записывается с заглавной 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). Имя у состояния может отсутствовать, т. е. оно является необязательным для некоторых состояний. В этом случае состояние является анонимным, и если на диаграмме состояний их несколько, то все они должны различаться между собой.

2.2.1. Начальное состояние

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). 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 (рисунок 2, а), из которого может только выходить стрелка, соответствующая переходу.

Рисунок 2 – Графическое изображение начального (а) и конечного (б) состояний на диаграмме состояний

На самом верхнем уровне представления объекта переход из начального состояния может быть помечен событием создания (инициализации) данного объекта. В противном случае переход никак не

помечается. Если этот переход не помечен, то он является первым переходом в следующее за ним состояние.

2.2.2. Конечное состояние

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосостояния). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. 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 (рисунок 1, б), в которую может только входить стрелка, соответствующая переходу.

2.3. Событие

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. Про события говорят, что они "происходят", при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

Семантика понятия «***событие***» фиксирует внимание на внешних проявлениях качественных изменений, происходящих при переходе моделируемого объекта из состояния в состояние. Например, при включении электрического переключателя происходит некоторое событие, в результате которого комната становится освещенной. После успешного ремонта компьютера также происходит немаловажное событие — восстановление его работоспособности.

В языке UML события играют роль стимулов, которые инициируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий. Имя события идентифицирует каждый отдельный переход на диаграмме состояний и может содержать строку текста, начинающуюся со строчной буквы. В этом случае принято считать переход ***триггерным***, т. е. таким, который специфицирует событие-триггер.

Если рядом со стрелкой перехода не указана никакая строка текста, то соответствующий переход является ***нетриггерным,*** и в этом случае из контекста диаграммы состояний должно быть ясно, после окончания какой деятельности он срабатывает. После имени события могут следовать круглые скобки для явного задания параметров соответствующего события-триггера. Если таких параметров нет, то список параметров со скобками может отсутствовать.

2.4. Сторожевое условие

Сторожевое условие (guard condition), если оно есть, всегда записывается в прямых скобках после события-триггера и представляет собой некоторое булевское выражение. Напомним, что булевское выражение должно принимать одно их двух взаимно исключающих значений: "истина" или "ложь". Семантика этого выражения должна явно следовать из контекста диаграммы состояний.

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. Если же сторожевое условие принимает значение "ложь", то переход не может сработать, и при отсутствии других переходов объект не может перейти в целевое состояние по этому переходу. Однако вычисление истинности сторожевого условия происходит только после возникновения ассоциированного с ним события-триггера, инициирующего соответствующий переход.

В общем случае из одного состояния может быть несколько переходов с одним и тем же событием-триггером. При этом никакие два сторожевых условия не должны одновременно принимать значение "истина". Каждое из сторожевых условий необходимо вычислять всякий раз при наступлении соответствующего события-триггера.

Примером события-триггера может служить разрыв телефонного соединения с провайдером Интернет-услуг после окончания загрузки электронной почты клиентской почтовой программой (при удаленном доступе к Интернету). В этом случае сторожевое условие есть не что иное, как ответ на вопрос: "Пуст ли почтовый ящик клиента на сервере провайдера?" В случае положительного ответа "истина", следует отключить соединение с провайдером, что и делает автоматически почтовая программа-клиент. В случае отрицательного ответа "ложь", следует оставаться в состоянии загрузки почты и не разрывать телефонное соединение.

Графически фрагмент логики моделирования почтовой программы может быть представлен в виде следующей диаграммы состояний (рисунок 3). Как можно заключить из контекста, в начальном состоянии программа не выполняется, хотя и имеется на компьютере пользователя. В момент ее включения происходит ее активизация. В этом состоянии программа может находиться неопределенно долго, пока пользователь ее не закроет, т. е. не выгрузит из оперативной памяти компьютера. После окончания активизации программа переходит в конечное состояние. В активном состоянии программы пользователь может читать сообщения электронной почты, создавать собственные послания и выполнять другие действия, не указанные явно на диаграмме.

Однако при необходимости получить новую почту, пользователь должен установить телефонное соединение с провайдером, что и показано явно на диаграмме верхним переходом. Другими словами, пользователь инициирует событие-триггер "установить телефонное соединение". В качестве параметра этого события выступает конкретный телефонный номер

модемного пула провайдера. Далее следует проверка сторожевого условия "телефонное соединение установлено", которое следует понимать как вопрос. Только в случае положительного ответа "да", т. е. "истина", происходит переход почтовой программы-клиента из состояния "активизация почтовой программы" в состояние "загрузка почты с сервера провайдера". В противном случае (линия занята, неверный ввод пароля, отключенный логин) никакой загрузки почты не произойдет, и программа останется в прежнем своем состоянии.

Рисунок 3 – Диаграмма состояний для моделирования почтовой программы-клиента

Второй триггерный переход на диаграмме инициирует автоматический разрыв телефонного соединения с провайдером после окончания загрузки почты на компьютер пользователя. В этом случае событие-триггер "закончить загрузку почты" происходит после проверки сторожевого условия "почтовый ящик на сервере пуст", которое также следует понимать в форме вопроса. При положительном ответе на этот вопрос (вся почта загружена или ее просто нет в ящике) почтовая программа прекращает загрузку почты и переходит в состояние активизации. В случае же отрицательного ответа загрузка почты будет продолжена.

3. Порядок выполнения работы

Для предметной области, проанализированной в лабораторной работе №1, составить диаграммы состояний для выбранных объектов (количество объектов должно быть не менее 5).

4. Содержание отчета

4.1. Цель работы.
4.2. Задание на работу.
4.3. Словесное описание предметной области.
4.4. Диаграммы состояния объектов.
4.5. Словесное описание диаграмм состояния.
4.6. Выводы по результатам работы.

5. Контрольные вопросы

5.1. Для чего используется диаграмма состояний?

5.2. Что понимается под состоянием?

5.3. Каким образом на диаграмме состояний обозначается начальное и конечное состояние?

5.4. Что понимается под событием?

5.5. Для чего используются сторожевые условия?

Лабораторная работа №4

«Исследование способов построения диаграмм видов деятельности»

1. Цель работы

Исследовать способы моделирования процесса выполнения операций. Изучить особенности использования состояний действия, переходов, дорожек и объектов.

2. Общие положения

2.1. Диаграммы видов деятельности

Для моделирования процесса выполнения операций в языке UML используются диаграммы видов деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме видов деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние происходит только при завершении этой операции в предыдущем состоянии. Графически диаграмма видов деятельности представляется в форме графа, вершинами которого являются состояния действия, а дугами — переходы от одного состояния действия к другому.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Именно они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий. Метамодель UML предоставляет для этого необходимые термины и семантику. Основным направлением использования диаграмм видов деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

В контексте языка UML ***деятельность*** (activity) представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

2.2. Состояние действия

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным. Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления.

Графически состояние действия изображается фигурой, напоминающей прямоугольник, боковые стороны которого заменены выпуклыми дугами (рисунок 1). Внутри этой фигуры записывается выражение действия (action-expression), которое должно быть уникальным в пределах одной диаграммы деятельности.

Рисунок 1 – Графическое изображение состояния действия

Действие может быть записано на естественном языке, некотором 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 (рисунок 1, а). Если же действие может быть представлено в некотором формальном виде, то целесообразно записать его на том языке программирования, на котором предполагается реализовывать конкретный проект (рисунок 1, б).

Иногда возникает необходимость представить на диаграмме деятельности некоторое сложное действие, которое, в свою очередь, состоит из нескольких более простых действий. В этом случае можно использовать специальное обозначение так называемого состояния ***поддеятельности*** (subactivity state). Такое состояние является графом деятельности и обозначается специальной пиктограммой в правом нижнем углу символа состояния действия (рисунок 2). Эта конструкция может применяться к любому элементу языка UML, который поддерживает "вложенность" своей структуры. При этом пиктограмма может быть дополнительно помечена типом вложенной структуры.

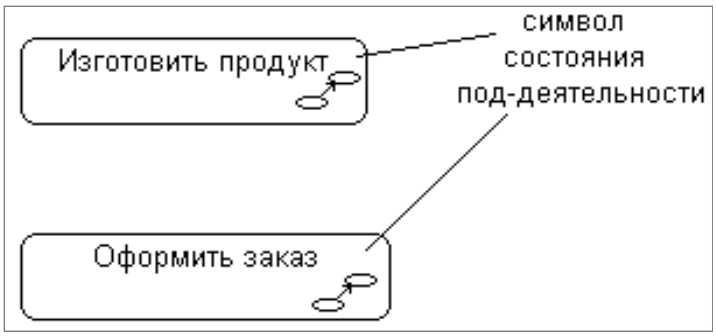

Рисунок 2 – Графическое изображение состояния поддеятельности

Каждая диаграмма видов деятельности должна иметь единственное начальное и единственное 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 видов деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное — в ее нижней части.

2.3. Переходы

При построении диаграммы видов деятельности используются только нетриггерные переходы, т. е. такие, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. Этот переход переводит деятельность в последующее состояние сразу, как только закончится действие в предыдущем состоянии. На диаграмме такой переход изображается сплошной линией со стрелкой.

Если из состояния действия выходит единственный переход, то он может быть никак не помечен. Если же таких переходов несколько, то сработать может только один из них. Именно в этом случае для каждого из таких переходов должно быть явно записано ***сторожевое условие*** в прямых скобках. При этом для всех выходящих из некоторого состояния переходов должно выполняться требование истинности только одного из них. Подобный случай встречается тогда, когда последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения некоторого промежуточного результата. Такая ситуация получила название ***ветвления***, а для ее обозначения применяется специальный символ.

Графически ***ветвление*** на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста. В этот 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 ветвления. Выходящих стрелок может быть две или более, но для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

В качестве примера рассмотрим фрагмент известного алгоритма нахождения корней квадратного уравнения. В общем случае после приведения уравнения второй степени к каноническому виду

$$a*x*x + b*x + c = 0$$

необходимо вычислить его дискриминант. Причем, в случае отрицательного дискриминанта уравнение не имеет решения на множестве действительных чисел, и дальнейшие вычисления должны быть прекращены. При неотрицательном дискриминанте уравнение имеет решение, корни которого могут быть получены на основе конкретной расчетной формулы.

Графически фрагмент процедуры вычисления корней квадратного уравнения может быть представлен в виде диаграммы деятельности с тремя состояниями действия и ветвлением (рисунок 3). Каждый из переходов, выходящих из состояния "Вычислить дискриминант", имеет сторожевое условие, определяющее единственную ветвь, по которой может быть продолжен процесс вычисления корней в зависимости от знака дискриминанта. Очевидно, что в случае его отрицательности, мы сразу попадаем в конечное состояние, тем самым завершая выполнение алгоритма в целом.

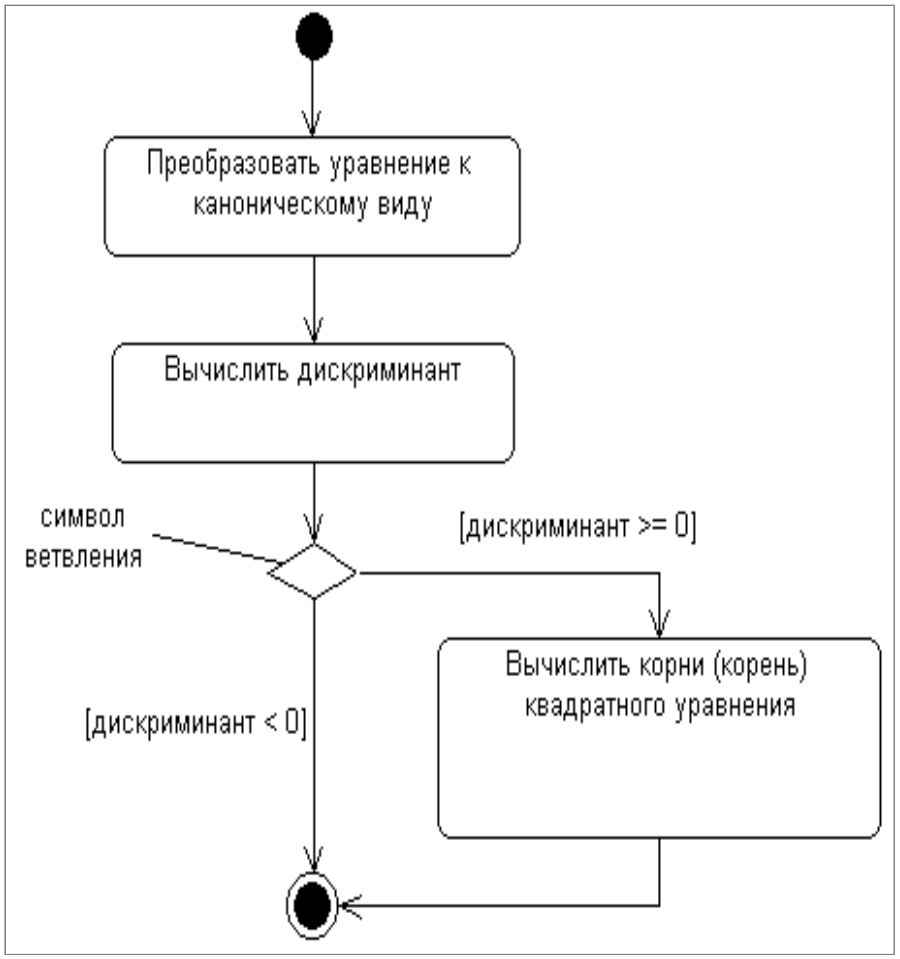

Рисунок 3 – Фрагмент диаграммы деятельности для алгоритма нахождения корней квадратного уравнения

2.4. Дорожки

Диаграммы видов 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. Действительно, деятельность любой компании (фирмы) также представляет собой не что иное, как совокупность отдельных действий, направленных на достижение требуемого результата. Однако применительно к бизнес-процессам желательно выполнение каждого действия ассоциировать с конкретным подразделением компании. В этом случае подразделение несет ответственность за реализацию отдельных действий, а сам бизнес-процесс представляется в виде переходов действий из одного подразделения к другому.

Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название «***дорожки***» (swimlanes). Имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму. При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании (рисунок 4).

Названия подразделений явно указываются в верхней части дорожки. Пересекать линию дорожки могут только переходы, которые в этом случае обозначают выход или вход потока управления в соответствующее подразделение компании. Порядок следования дорожек не несет какой-либо семантической информации и определяется соображениями удобства.

В качестве примера рассмотрим фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов по телефону. Подразделениями компании 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 данном случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое из подразделений торговой компании должно выполнять то или иное действие (рисунок 5).

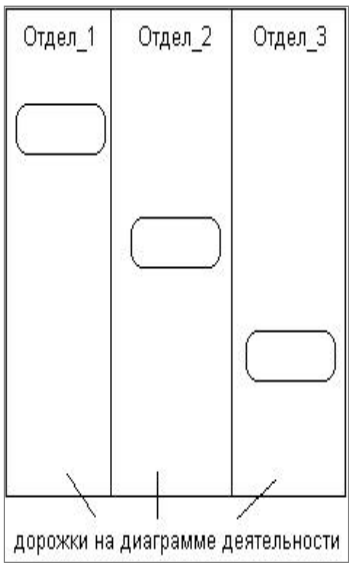

Рисунок 4 – Вариант диаграммы деятельности с дорожками

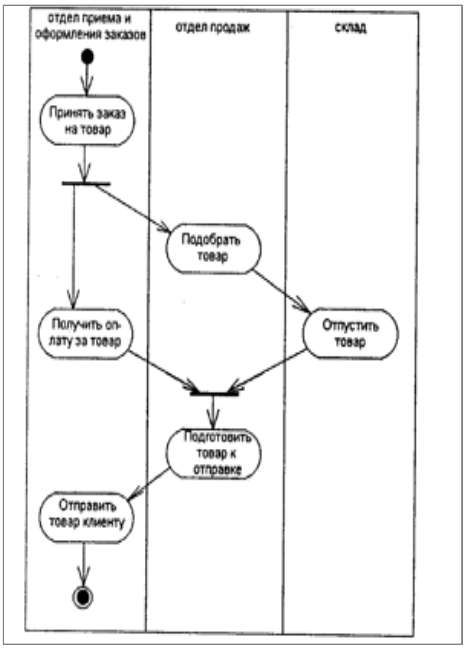

Рисунок 5 – Фрагмент диаграммы деятельности для торговой компании

Из указанной диаграммы видов деятельности сразу видно, что 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 подбору товара в отделе продаж (модель товара, размеры, цвет, год выпуска и пр.). По окончании этой работы инициируется действие по отпуску товара со склада. Однако подготовка товара к отправке в торговом отделе начинается только после того, как будет получена оплата за товар от клиента и товар будет отпущен со склада (переход-соединение). Только после этого товар отправляется клиенту, переходя в его собственность.

2.5. Объекты

В общем случае действия на диаграмме видов деятельности выполняются над теми или иными объектами. Эти объекты либо инициируют выполнение действий, либо определяют некоторый результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности к другому. Поскольку в таком ракурсе объекты играют определенную роль в понимании процесса деятельности, иногда возникает необходимость явно указать их на диаграмме видов деятельности.

Для графического представления объектов, используются прямоугольник класса, с тем отличием, что имя объекта подчеркивается. Далее после имени может указываться характеристика состояния объекта в прямых скобках. Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой. Соответствующая зависимость определяет состояние конкретного объекта после выполнения предшествующего действия.

На диаграмме видов деятельности с дорожками расположение объекта может иметь некоторый 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 готовностью некоторого документа (объект в некотором состоянии). Если же объект целиком расположен внутри дорожки, то и состояние этого объекта целиком определяется действиями данной дорожки.

Возвращаясь к предыдущему примеру с торговой компанией, можно заметить, что центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до звонка от клиента заказ как объект отсутствует и возникает лишь после такого звонка. Однако этот заказ еще не заполнен до конца, поскольку требуется еще подобрать конкретный товар в отделе продаж. После его подготовки он передается на склад, где вместе с отпуском товара заказ окончательно дооформляется. Наконец, после получения подтверждения об оплате товара эта информация заносится в заказ, и он считается выполненным и закрытым. Данная информация может быть представлена графически в виде модифицированного варианта диаграммы деятельности этой же торговой компании (рисунок 6).

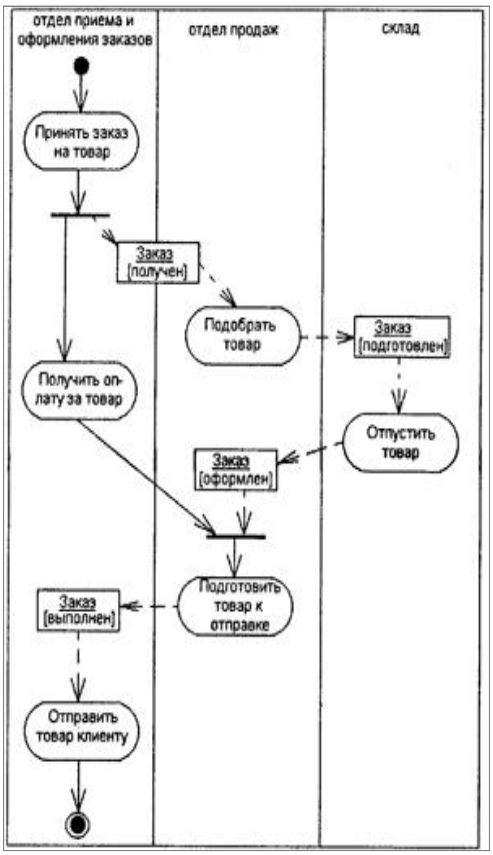

Рисунок 6 – Фрагмент диаграммы деятельности торговой компании с объектом-заказом

3. Порядок выполнения работы

Для предметной области, проанализированной в лабораторной работе №1, составить диаграммы видов деятельности. Использовать дорожки для иллюстрации взаимодействия объектов.

4. Содержание отчета

4.1. Цель работы.
4.2. Задание на работу.
4.3. Словесное описание предметной области.
4.4. Диаграммы видов деятельности.
4.5. Словесное описание диаграмм видов деятельности.
4.6. Выводы по результатам работы.

5. Контрольные вопросы

5.1. Для каких целей используется диаграмма видов деятельности?
5.2. Что такое состояние действия?
5.3. Расскажите о переходах на диаграммах видов деятельности.
5.4. Что понимается под дорожкой на диаграммах видов деятельности?

5.5. Каким образом и для чего используются объекты на диаграммах видов деятельности?

5.6. Расскажите об общих этапах архитектурного проектирования.

5.7. Структурирование системы: расскажите о модели репозитория и модели клиент-сервер.

5.8. Расскажите о моделях управления при архитектурном проектировании.

Лабораторная работа №5
«Исследование способов построения диаграмм последовательностей»

1. Цель работы

Исследовать способы представления временных особенностей передачи и приема сообщений между объектами. Изучить способы представления объектов, сообщений и времени на диаграммах последовательностей.

2. Общие положения

Одной из характерных особенностей систем различной природы и назначения является взаимодействие между собой отдельных элементов, из которых образованы эти системы. Речь идет о том, что различные составные элементы систем не существуют изолированно, а оказывают определенное влияние друг на друга, что и отличает систему как целостное образование от простой совокупности элементов.

В языке UML взаимодействие элементов рассматривается в информационном аспекте их коммуникации, т. е. взаимодействующие объекты обмениваются между собой некоторой информацией. При этом информация принимает форму законченных сообщений. Другими словами, хотя сообщение и имеет информационное содержание, оно приобретает дополнительное свойство оказывать направленное влияние на своего получателя.

Взаимодействия объектов можно рассматривать во времени, тогда для представления временных особенностей передачи и приема сообщений между объектами используется ***диаграмма последовательности***.

2.1. Объекты

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения. Одно — слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни (рисунок 1). Внутри прямоугольника записываются имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта, который, как известно, представляет собой экземпляр класса.

Не исключается ситуация, когда имя объекта может отсутствовать на диаграмме последовательности. В этом случае указывается только имя класса, а сам объект считается анонимным.

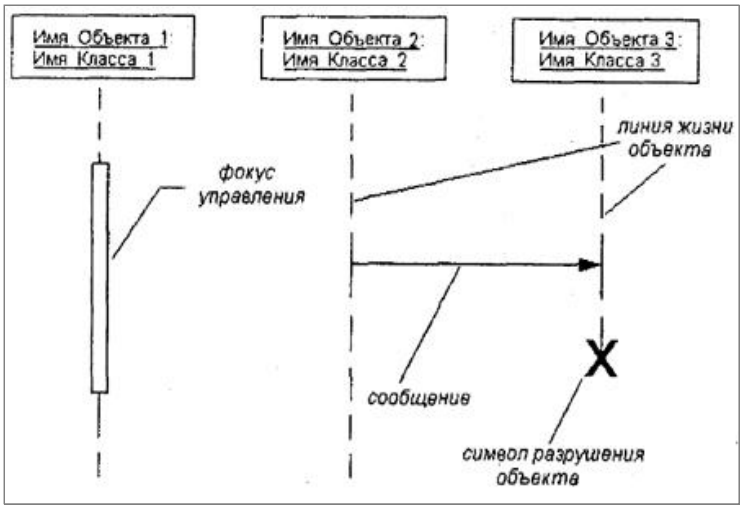

Рисунок 1 – Различные графические примитивы диаграммы последовательности

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия (объект 1 на рисунке 1). Правее изображается 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.

Второе измерение диаграммы последовательности — вертикальная временная ось, направленная сверху вниз. Начальному моменту времени соответствует самая верхняя часть диаграммы. При этом взаимодействия объектов реализуются посредством сообщений, которые посылаются одними объектами другим. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения и также образуют порядок по времени своего возникновения. Другими словами, сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа "раньше-позже".

2.2. Линия жизни объекта

Линия жизни объекта (object lifeline) 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, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна

продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней (объекты 1 и 2 на рисунке 1).

Отдельные объекты, выполнив свое задание в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы "X" (объект 3 на рисунке 1). Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Вовсе не обязательно создавать все объекты в начальный момент времени. 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 ее части, которая соответствует моменту создания объекта (объект 6 на рисунке 2). При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Очевидно, объект обязательно создается со своей линией жизни и, возможно, с ***фокусом управления***.

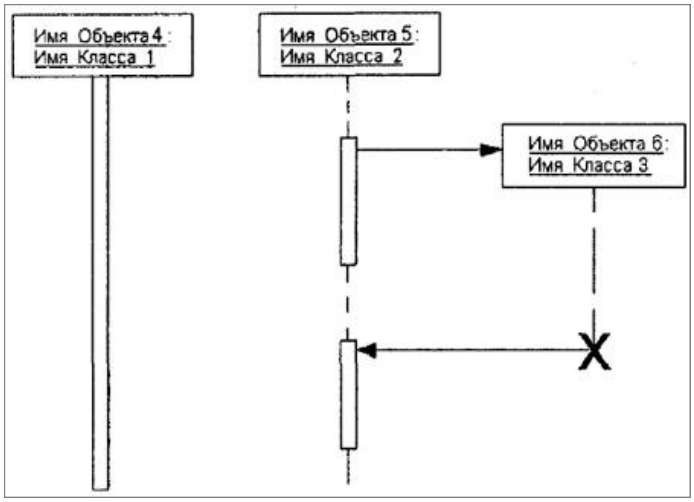

Рисунок 2 – Графическое изображение различных вариантов линий жизни и фокусов управления объектов

2.3. Фокус 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. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название ***фокуса управления*** (focus of control). Фокус управления изображается в форме вытянутого узкого прямоугольника (см. объект 1 на рисунок 1), 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 4 на рисунке 2), если на всем ее протяжении он является активным.

С другой стороны, периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта имеются несколько фокусов управления (объект 5 на рисунке 2). Важно понимать, что получить фокус управления может только существующий объект, у которого в этот момент имеется линия жизни. Если же некоторый объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него лишь может быть создан другой экземпляр этого же класса, который, строго говоря, будет являться другим объектом.

В отдельных случаях инициатором взаимодействия в системе может быть актер или внешний пользователь. В этом случае актер изображается на диаграмме последовательности самым первым объектом слева со своим фокусом управления (рисунке 3). Чаще всего актер и его фокус управления будут существовать в системе постоянно, отмечая характерную для пользователя активность в инициировании взаимодействий с системой. При этом сам актер может иметь собственное имя либо оставаться анонимным.

Иногда некоторый объект может инициировать рекурсивное взаимодействие с самим собой. Речь идет о том, что наличие во многих языках программирования специальных средств построения рекурсивных процедур требует визуализации соответствующих понятий в форме графических примитивов. На диаграмме последовательности рекурсия обозначается небольшим прямоугольником, присоединенным к правой стороне фокуса управления того объекта, для которого изображается это рекурсивное взаимодействие (объект 7 на рисунке 3).

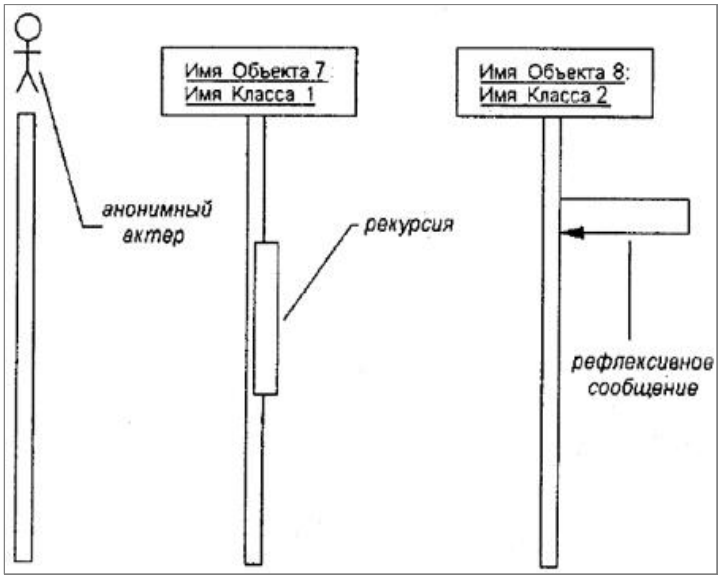

Рисунок 3 – Графическое изображение актера, рекурсии и рефлексивного сообщения на диаграмме последовательности

2.4. Сообщения

Как было отмечено выше, цель взаимодействия в контексте языка UML заключается в том, чтобы специфицировать коммуникацию между множеством взаимодействующих объектов. 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.

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.

В таком контексте каждое сообщение имеет направление от объекта, который инициирует и отправляет сообщение, к объекту, который его получает. 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.

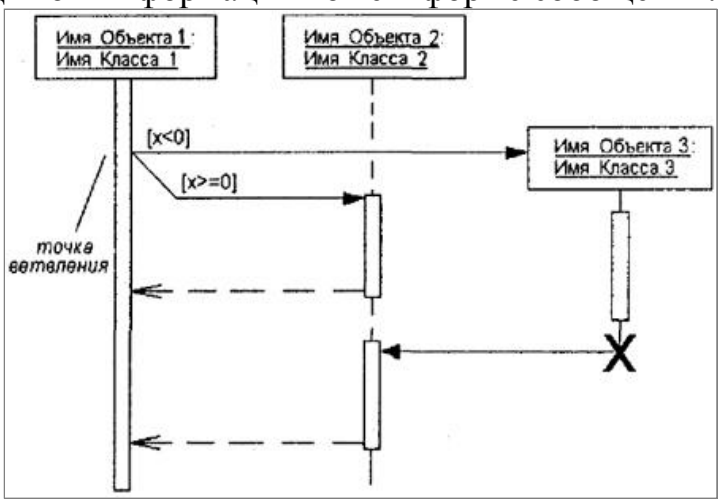

Рисунок 4 – Графическое изображение различных видов сообщений между объектами на диаграмме последовательности

В языке UML могут встречаться несколько разновидностей сообщений, каждое из которых имеет свое графическое изображение (рисунок 4).

- Первая разновидность сообщения является наиболее распространенной и используется для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления. Начало этой стрелки всегда соприкасается с фокусом управления или линией жизни 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 зачастую получает и фокус управления, становясь активным.

- Вторая разновидность сообщения используется для обозначения простого (не вложенного) потока управления. Каждая такая стрелка указывает на прогресс одного шага потока. При этом соответствующие сообщения обычно являются асинхронными, т. е. могут возникать в произвольные моменты времени. Передача такого сообщения обычно сопровождается получением фокуса управления объектом, его принявшим.

- Третья разновидность явно обозначает асинхронное сообщение между двумя объектами в некоторой процедурной последовательности. Примером такого сообщения может служить прерывание операции при возникновении исключительной ситуации. В этом случае информация о такой ситуации передается вызывающему объекту для продолжения процесса дальнейшего взаимодействия.

- Наконец, последняя разновидность сообщения используется для возврата из вызова процедуры. Примером может служить простое сообщение о завершении некоторых 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 остаются без изменения. Если же это предположение не может быть признано справедливым, то стрелка сообщения изображается под некоторым наклоном, так чтобы конец стрелки располагался ниже ее начала.

В отдельных случаях объект может посылать сообщения самому себе, инициируя так называемые рефлексивные сообщения (объект 8 на рисунке 3). Такие сообщения изображаются прямоугольником со стрелкой, начало и конец которой совпадают. Подобные ситуации возникают, например, при обработке нажатий на клавиши клавиатуры при вводе текста в редактируемый документ, при наборе цифр номера телефона абонента.

Таким образом, в языке UML каждое сообщение ассоциируется с некоторым действием, которое должно быть выполнено принявшим его объектом. При этом действие может иметь некоторые аргументы или параметры, в зависимости от конкретных значений которых может быть получен различный результат. Соответствующие параметры 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.

3. Порядок выполнения работы

Для предметной области, проанализированной в лабораторной работе №1, составить диаграммы последовательностей.

4. Содержание отчета

4.1. Цель работы.
4.2. Задание на работу.
4.3. Словесное описание предметной области.
4.4. Диаграммы последовательностей.
4.5. Словесное описание диаграмм последовательностей.
4.6. Выводы по результатам работы.

5. Контрольные вопросы

5.1. Для чего используются диаграммы последовательностей?
5.2. Каким образом отображаются объекты на диаграммах последовательностей?
5.3. Что такое линия жизни?
5.4. Какие бывают виды сообщений? Опишите каждый из них.
5.5. Какие основные условия должны выполняться для успешного проектирования и разработки ПО с повторно используемыми компонентами?
5.6. Назовите основные проблемы при разработке ПО с повторно используемыми компонентами.
5.7. Назовите и охарактеризуйте основные уровни абстракции компонентов.
5.8. Какими особенностями должен обладать повторно используемый компонент?

Библиографический список

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ - 2-е изд. / Г. Буч.- М.: Издательство Бином; СПб.: Невский диалект, 1999.

2. Буч Г. Язык UML. Руководство пользователя - 2-е изд. / Г. Буч, Д. Рамбо, И. Якобсон.- М.:ДМК Пресс, 2007.- 496 с.

3. Фаулер М. UML. Основы / М. Фаулер, К. Скотт.- СПб.: Символ-Плюс, 2002.- 192 с.

4. Освой самостоятельно UML за 24 часа, 2-е издание. : Пер. с англ. — М.: Издательский дом "Вильяме", 2002. — 352 с.

ПРИЛОЖЕНИЕ А

Система “Корпоративная библиотека”.

Система предназначена для автоматизации работы служащих корпоративной библиотеки и предоставления ряда сервисов клиентам (читателям).

Система предполагает наличие разных отделов, и должностей работников (библиотекарь, управляющий, заведующий отделом), а также групп пользователей, разделенных по уровням доступа к материалам библиотеки.

Система должна содержать следующие основные возможности: для работников библиотеки — заказ книги на покупку, списание книги, добавление книги в каталог, просмотр каталога книг, удаление книги из каталога, выдача и возврат книг от клиентов; для читателей — просмотр каталога книг (с возможностью сортировки и поиска по различным полям), контроль за взятыми книгами и сроками их возврата. Также система должна контролировать возможность выдачи книги читателю в соответствии с его уровнем допуска.

Система "Обслуживание клиента в банке".

Система предназначена для предоставления услуг банка клиентам и автоматизации работы служащих банка. Необходимо предусмотреть несколько отделов банка, занимающиеся предоставлением различных услуг, таких как страхование, предоставление кредита, открытие депозита, предоставление кредитных, дебетовых и платежных карт на основе составления соответствующего договора с клиентом, а также осуществление денежных переводов.

Также система должна автоматизировать работу кассира. Кассир взаимодействует с клиентом, предоставляя ему услуги купли-продажи валюты, возможность кредитного взноса, пополнение депозита через кассу банка, прием и выдача наличных для различных целей.

Клиент может иметь множество кредитов, депозитов и счётов в банке, а так же свой сейф, который арендует в процессе взаимодействия с системой.

Система должна предоставлять клиенту все необходимую ему информацию.

Система “Обучение студента в университете”.

Система должна автоматизировать работу разных подразделений университета, а также обеспечить студента необходимой информацией. Основными сторонами учебы студента в университете должны являться следующие: учеба, быт, отдых.

Система должна содержать следующие основные возможности: обеспечивать доступ студента к расписанию занятий, а также расписанию

необходимых ему преподавателей; просмотр оценок и задолженностей с возможностью выбора конкретного предмета и вида контроля знаний (оценки по лабораторным работам, контрольным, РГЗ, модулей, зачетов, экзаменов), а также посещений различных занятий (лабораторные работы, практика, лекции, экзамены, ликвидация задолженностей); контроль за взятыми в библиотеке книгами и методическими указаниями с возможностью напоминания о сроках, а также просмотра списка рекомендуемой литературы по выбранным предметам; доступ к информации связанной с проживанием студента в общежитии (при необходимости); предоставление новостей кафедр, деканата, профсоюза и студактива; контроль состояния оплаты за учебу (при необходимости).

Система “Учет успеваемости студента в университете”.

Система должна обеспечивать автоматизацию основных подразделений университета связанных с обучением студента в университете.

Необходимо предусмотреть различных пользователей системой: студент, преподаватели с различными должностями (ассистент, ведущий преподаватель), методист деканата, декан факультета.

Система должна содержать следующие основные возможности: для студентов — просмотр оценок и задолженностей с возможностью выбора конкретного предмета и вида контроля знаний (оценки по лабораторным работам, контрольным, РГЗ, модулей, зачетов, экзаменов), а также посещений различных занятий (лабораторные работы, практика, лекции, экзамены, ликвидация задолженностей); для преподавателей — выставление оценок и посещаемости студентов в тех предметах, которые они ведут; для методиста деканата — функции составления отчетов по успеваемости студентов, создание ведомостей на сдачу экзаменов; для декана факультета — просмотр оценок по группе, по потоку и по предмету, функции допуска студентов к сдаче экзаменов, функции представления студентов на отчисление.

Система “Интернет-магазин по продаже компьютерной техники”.

Система предназначена для автоматизации работы различных отделов интернет-магазина и предоставления услуг клиентам. Система должна предусматривать различные роли пользователей — клиент, продавец, администратор.

Система должна содержать следующие основные возможности: для клиентов фирмы — поиск, сортировка, фильтрация товаров в каталоге; покупка товара с возможностью оплаты on-line; оформление доставки товара на указанный адрес; функция задания вопросов продавцу; для администратора — добавление, удаление, изменение параметров (цена, количество) товаров в каталоге; для продавца — подтверждение заказа товара, подтверждение оплаты товара, комплектация заказа; ответ на вопросы клиентов; добавление описание товаров, комментариев.

Система “Работа банкомата”.

Система предназначена для управления работой банкомата некоторого банка. Основной функцией системы должно являться удовлетворение запросов клиентов, а также работников банка, в частности обслуживающего персонала.

Таким образом, для клиента банкомат должен выполнять следующие основные функции: пополнение текущего счета клиента наличными, снятие наличных с текущего счета клиента, осуществление оплаты за различные услуги (коммунальные и т. д.); просмотр текущего состояния счета клиента; информирование клиента об изменении состояния счета клиента. Для работников банка система должна реализовывать следующие сервисы: сигнализировать соответствующие службы банка о некоторых событиях — переполнение банкомата наличными, нехватка наличных средств в банкомате; факт забытия карты в банкомате; несанкционированный доступ к счету; попытка взлома.

Система “Магазин по продаже компьютерной техники”.

Система предназначена для автоматизации работы магазина, в котором необходимо предусмотреть работу нескольких подразделений.

В системе должны быть реализованы следующие функции: для руководства магазина: заказ товаров у поставщиков, просмотр статистики по продажам за разные периоды и различные виды товаров; для работников склада: прием товаров на баланс, учет товаров на складе (просмотр наличия товара на складе, его количества), формирование отчета по изменению загруженности склада; для кассиров — принять деньги за товар, выдать деньги в случае возврата товара, сформировать чек для покупателя; для продавцов — формирование накладной для покупателя.

Система “Офис локальной сети”.

Система предназначена для автоматизации работы компании предоставляющей доступ к услугам локальной сети и Интернет для различных пользователей, а также предоставления некоторых услуг клиентам компании. Система может содержать следующих основных пользователей: руководство локальной сети, администраторы локальной сети, инженеры, клиенты.

В системе должны быть реализованы следующие функции: для клиентов — заказать подключение к локальной сети (для будущих клиентов), просмотр и изменение текущего тарифного плана, просмотр текущего баланса, а также оформление заявки на неисправность в работе сети; для инженера — прием заявки на подключение нового клиента, отключение клиента от локальной сети, обработка заявки об неисправности в работе сети; для администратора — добавление вновь подключенных клиентов базу, блокировка и отмена блокировки доступа клиентов к различным сервисам; для руководства сети — просмотр базы данных клиентов и получение

различной статистической информации (количество подключенных клиентов за определенный период, доход за определенный период, и т.д.).

Система “Офис страховой фирмы”.

Система предназначена для автоматизации процесса документооборота компании предоставляющей различные услуги страхования для физических лиц. Основными функциями системы должны являться ведение клиентской базы, составление договоров с клиентами на предоставление услуг страхования (страхование здоровья и жизни, имущества, автогражданского страхования и др.)

Система “Редакция журнала и газеты”.

Система предназначена для автоматизации процесса документооборота в редакции некоторого журнала и газеты. Система должна предусматривать наличие различных отделов редакции: журналисты, дизайнеры, фотографы, редакторы, верстальщик.

Система “Строительная фирма”.

Система предназначена для автоматизации работы различных отделов фирмы и предоставления услуг клиентам.

Основными функциями системы являются следующие: отслеживание этапов строительства; контроль за выполнением работ строительными подразделениями (бригадами) их объем и сроки завершения; заказ стройматериалов, учет их использования.