Министерство науки и высшего образования Российской Федерации

Севастопольский государственный университет

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНОЙ РАБОТЕ № 2
«ДОКУМЕНТИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ, ОБЪЕКТА И МЕТОДА ИССЛЕДОВАНИЙ НА ЯЗЫКЕ UML»

по дисциплине

«ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В НАУКЕ И ОБРАЗОВАНИИ»

для аспирантов дневной и заочной форм обучения

Севастополь, 2018

Методические указания к лабораторной работе № 2 «Документирование предметной области, объекта и метода исследований на языке UML» по дисциплине «Информационные технологии в науке и образовании» для аспирантов дневной и заочной форм обучения. Сост.: Мащенко Е.Н., Николаева Ю.П. - Севастополь, изд-во СевГУ, 2018. - 48 С.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Лабораторная работа выполняется на тему «Документирование результатов учебно-исследовательской работы с использованием языка UML».

Язык UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем.

Язык UML является достаточно строгим и мощным средством моделирования, которое может быть эффективно использовано для построения концептуальных, логических и графических моделей сложных систем различного целевого назначения.

Этот язык вобрал в себя наилучшие качества и опыт методов программной инженерии, которые с успехом использовались на протяжении последних лет при моделировании сложных систем.

Суть лабораторной работы заключается в разработке модели предметной области, содержательного описания вариантов использования и ряда диаграмм UML, описывающих программную систему, являющуюся результатом учебно-исследовательской работы.

Задание на лабораторную работу сформулировано в разделе 1 настоящих указаний. Лабораторная работа выполняется по индивидуальным вариантам.

В разделе 2 приводятся теоретические сведения по языку UML и примеры диаграмм.

В разделе 3 содержатся краткие методические указания по выполнению задания.

Разработка диаграмм UML может производиться вручную либо с использованием специализированных программных систем (Visio, Rational Rose и др.), поддерживающих язык UML для разработки и графического представления результатов проектирования.

1. ОБЩАЯ ФОРМУЛИРОВКА ЗАДАНИЯ НА ЛАБОРАТОРНУЮ РАБОТУ

Вариант задания представляет собой словесное описание структуры и алгоритмов функционирования некоторой системы, а также список целей функционирования и критериев качества - в качестве задания могут использоваться результаты выполнения лабораторной работы № 1 «Проведение системного анализа предметной области, объекта и метода исследований».

В рамках выполнения лабораторной работы необходимо:

1) Определить варианты использования (описать на естественном языке и построить диаграммы).
2) Создать модель предметной области (построить и описать диаграмму концептуальных классов).
3) Построить и описать диаграммы взаимодействия;
3) Построить и описать диаграмму классов проектирования (для программных систем).
5) Для выбранной подсистемы построить и описать диаграмму состояний.
6) Для выбранного метода построить и описать диаграмму видов деятельности.

2. ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ ПО ЯЗЫКУ UML

2.1. Общие сведения

UML (Unified Modeling Language, Унифицированный язык моделирования) – это:

1) общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования ПО, бизнес-процессов и других систем;
2) юридический и фактический стандарт системы обозначений для построения диаграмм в рамках объектно-ориентированного моделирования и проектирования программного обеспечения.

UML представляет собой универсальный язык, позволяющий одновременно с анализом создавать документацию для проектирования сложных иерархических систем с последующим воплощением проекта в виде программной системы с использованием языков программирования, поддерживающих объектно-ориентированный подход (ООП).

Создан в 1994 г. Гради Бучем (Grady Booch), Джимом Румбахом (Jim Rumbaugh), Айваром Якобсоном путем обьединения их популярных методов: метода Буча, OMT(Object Modeling Techique) и OOSE(Object-Oriented Software Engineering).

Стандарты: 1997 г – UML1, 2003 г - UML2.

Конструктивное использование UML основывается на понимании общих принципов моделирования сложных систем и особенностей объектно-ориентированных анализа и проектирования в частности.

Одним из основных принципов построения моделей сложных систем является принцип абстрагирования, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения.

Другим принципом построения моделей сложных систем является принцип многомодельности, который утверждает, что никакая единственная модель не может с достаточной степенью адекватности описывать различные аспекты сложной системы. Применительно к методологии объектно-ориентированного анализа и проектирования (ООАП) это означает, что достаточно полная модель сложной системы допускает некоторое число взаимосвязанных представлений (views), каждое из которых адекватно отражает некоторый аспект поведения или структуры системы.

Ещё одним принципом построения моделей сложных систем является принцип иерархического построения моделей. Этот принцип предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или детализации в рамках фиксированных представлений. При этом исходная

или первоначальная модель сложной системы имеет наиболее общее представление (метапредставление).

Таким образом, процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня.

UML предназначен для решения следующих задач:

1) обеспечения пользователей легко воспринимаемым и выразительным языком визуального моделирования, специально предназначенного для разработки и документирования моделей сложных систем самого различного целевого назначения;

2) предоставления возможности расширения и специализации языка для более точного представления моделей систем в конкретной предметной области;

3) обеспечения независимости спецификаций моделей от конкретных языков программирования и инструментальных средств проектирования программных систем;

4) обеспечения семантического базиса, отражающего особенности ООАП.

В рамках UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм. В терминах UML версии 2.2 определены 14 видов диаграмм, разделенных на две категории: «поведения» и «структурные».

Диаграммы поведения (behavior diagram):

1) диаграмма вариантов использования или прецедентов (use case diagram);
2) диаграмма деятельности (activity diagram);
3) диаграмма состояний (state machine diagram);
4) диаграммы взаимодействия (interaction diagrams):
    - 4.1) диаграмма последовательностей (sequence diagram);
    - 4.2) диаграмма коммуникации (communication diagram);
    - 4.3) диаграмма обзора взаимодействия (Interaction overview diagram);
    - 4.4) диаграмма синхронизации (Timing diagram).

Структурные диаграммы:

5) диаграмма классов (class diagram);
6) диаграмма компонентов (component diagram);
7) диаграмма кооперации (Композитной/составной структуры) (collaboration diagram);
8) диаграмма развёртывания (deployment diagram);
9) диаграмма объектов (object diagram);
10) диаграмма пакетов (package diagram);
11) диаграмма профилей (profile diagram).

В лабораторном проекте предполагается построение диаграмм 1), 2), 3), 4.1), 5).

При графическом изображении диаграмм следует придерживаться следующих основных рекомендаций.

1. Каждая диаграмма должна быть законченным представлением соответствующего фрагмента моделируемой предметной области. В процессе разработки диаграммы необходимо учесть все сущности, важные с точки зрения контекста данной модели и диаграммы. Отсутствие тех или иных существенных элементов на диаграмме служит признаком неполноты модели.
2. Все сущности на диаграмме модели должны быть одного концептуального уровня. В случае достаточно сложных моделей систем желательно придерживаться стратегии последовательного уточнения или детализации отдельных диаграмм.
3. Вся информация о сущностях должна быть явно представлена на диаграммах.
4. Диаграммы не должны содержать противоречивой информации.
5. Диаграммы не следует перегружать текстовой информацией.
6. Каждая диаграмма должна быть самодостаточной для правильной интерпретации всех её элементов и понимания семантики всех используемых графических символов.
7. Количество типов диаграмм для конкретной модели приложения не является строго фиксированным. Некоторые типы диаграмм могут отсутствовать в проекте системы. Перечень диаграмм зависит от специфики конкретной информационной системы.

2.2. Обозначения и примеры диаграмм UML

2.2.1. Диаграмма вариантов использования

Диаграмма *вариантов использования* описывает функциональное назначение системы. Построение диаграммы вариантов использования является самым первым этапом процесса ООАП, цель которого – представить совокупность *требований* к поведению проектируемой системы.

Требование можно определить как «описание того, что должно быть реализовано». Требования указывают, *что* должно быть построено, но не определяют, *как* именно. Существуют два основных типа требований:

- функциональные требования – какие действия должна выполнять система;
- нефункциональные требования – особые свойства системы или накладываемые на нее ограничения.

Для записи требований используется ключевое слово «должен». Пример функционального требования: «Система должна предоставлять авторизованному пользователю доступ к его персональной информации».Пример нефункционального требования: «Система должна быть написана на C++».

Моделирование вариантов использования (прецедентов) – это форма выработки требований. Можно считать варианты использования способом записи требований.

Разработка диаграммы вариантов использования преследует следующие цели:

1) определение общих границ и контекста моделируемой предметной области на начальных этапах проектирования системы;
2) формулирование общих требований к функциональному поведению проектируемой системы;
3) разработка исходной концептуальной модели системы для её последующей детализации в виде логической и физической моделей;
4) подготовка исходной документации для взаимодействия разработчиков системы с её заказчиками и пользователями.

Стадии моделирования вариантов использования:

1. Устанавливаются границы системы.
2. Выявляются актеры (actor).
3. Выявляются варианты использования (use case).
   - 3.1. Определяются варианты использования;
   - 3.2. Устанавливаются основные и альтернативные потоки событий (успешные и неуспешные сценарии поведения системы).
4. Проводится описание (специфицирование) вариантов использования в произвольной либо табличной форме.
5. Реализуется графическая нотация (диаграмма) вариантов использования.

Результатом является модель вариантов использования, которая включает следующие компоненты:

- *Граница системы* – прямоугольник, отделяющий систему от ее окружения. В UML 2 границу системы называют контекстом системы (subject).
- *Актеры.*
- *Варианты использования.*
- *Отношения* – значимые отношения между актерами и вариантами использования.

В самом общем случае диаграмма вариантов использования представляет собой граф специального вида, который является графической нотацией для представления конкретных вариантов использования, актёров и отношений между ними.

Актёром (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 ис-

точником воздействия на моделируемую систему так, как определит разработчик.

Обычно актеры выявляются из описания задачи системы. Для выявления актеров может быть использована следующая группа вопросов:

4. Кто или что использует систему?
5. Кто устанавливает систему?
6. Кто или что запускает и выключает систему?
7. Кто осуществляет поддержку и обслуживание системы?
8. Какие системы взаимодействуют с данной системой?
9. Кто снабжает систему информацией, использует информацию, и получает информацию из системы?
10. Происходит ли что-нибудь в точно установленное время?
11. Выступает ли какой-либо участник системы в нескольких ролях ?
12. Выступают ли различные участники в одной роли?

Вариант использования служит для описания сервисов, которые система предоставляет актёру. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актёров с системой.

Вариант использования *всегда* инициируется актером. Вариант использования *всегда* описывается с точки зрения актера.

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

Цель варианта использования заключается в том, чтобы определить законченный аспект или фрагмент поведения некоторой сущности без раскрытия её внутренней структуры. В качестве такой сущности может выступать система или любой элемент модели, который обладает собственным поведением.

Каждый вариант использования соответствует отдельному сервису, который предоставляет моделируемая сущность по запросу актера, то есть определяет способ применения этой сущности. Сервис, который инициализируется по запросу актера, представляет собой законченную неделимую последовательность действий. Это означает, что после того как система закончит обработку запроса, она должна возвратиться в исходное состояние, чтобы быть готовой к выполнению следующих запросов.

Варианты использования могут применяться как для спецификации внешних требований к проектируемой системе, так и для спецификации функционального поведения уже существующей системы. Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы. Кроме этого, варианты использования неявно устанавливают требования, определяющие, как актеры должны взаимодействовать с системой, чтобы иметь возможность корректно работать с предоставляемыми сервисами.

В описании варианта использования указывается, что должна делать система при его выполнении. Все функциональные возможности системы определяются именно набором вариантов использования, каждый из которых представляет определённый поток событий.

В UML предусмотрены следующие стандартные виды взаимодействий (отношений) между актёрами и вариантами использования:

1) *отношение ассоциации* (association relationship);
2) *отношение расширения* (extend relationship);
3) *отношение обобщения* (generalization relationship);
4) *отношение включения* (include relationship).

Отношение ассоциации.

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

На диаграмме вариантов использования отношение ассоциации обозначается *сплошной линией* между актером и вариантом использования. Эта линия может иметь условные обозначения, такие как имя и кратность. Пример отношения ассоциации приведен на рисунке 2.1.1.

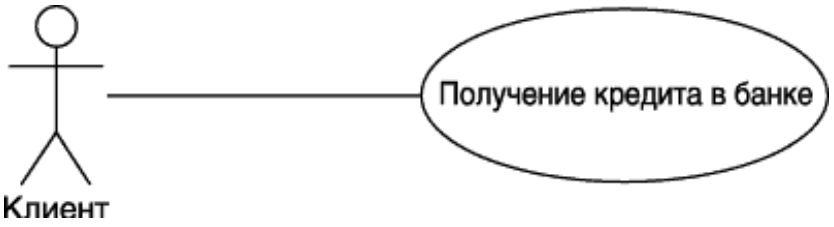


Рисунок 2.1.1 - Пример отношения ассоциации

Кратность (multiplicity) ассоциации указывается рядом с обозначением компонента диаграммы, который является участником данной ассоциации, и характеризует количество экземпляров данного компонента, которые могут выступать в качестве элементов данной ассоциации. Применительно к диаграммам вариантов использования кратность имеет специальное обозначение в форме одной или нескольких цифр и символа звездочка.

Для диаграмм вариантов использования наиболее распространенными являются четыре основные формы записи кратности отношения ассоциации:

- Целое неотрицательное число (включая 0). Предназначено для указания кратности, которая является строго фиксированной для элемента соответствующей ассоциации. В этом случае количество экземпляров актеров или вариантов использования, которые могут выступать в качестве элементов отношения ассоциации, в точности равно указанному числу;
- Два целых неотрицательных числа, разделенные двумя точками. Данная запись соответствует нотации для множества или интервала целых чисел, которая применяется в некоторых языках программирования для обозначения границ массива элементов. Эту запись следует понимать как множест-

во целых неотрицательных чисел, следующих в последовательно возрастающем порядке;

- Два символа, разделенные двумя точками. При этом первый из них является целым неотрицательным числом или 0, а второй - специальным символом «*», который обозначает произвольное конечное целое неотрицательное число, значение которого неизвестно на момент задания соответствующего отношения ассоциации;
- Единственный символ «*», который является сокращением записи интервала «0..*».

Если кратность отношения ассоциации не указана, то, по умолчанию, принимается значение равное 1.

Отношение расширения.

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

Отношение расширения является направленным и указывает, что применительно к отдельным примерам некоторого варианта использования должны быть выполнены конкретные условия, определенные для расширения данного варианта использования. Так, если имеет место отношение расширения от варианта использования А к варианту использования В, то это означает, что свойства экземпляра варианта использования В могут быть дополнены благодаря наличию свойств у расширенного варианта использования А.

Отношение расширения между вариантами использования обозначается пунктирной линией со стрелкой (вариант отношения зависимости), направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом «extend» (расширяет). . Пример отношения расширения приведен на рисунке 2.1.2.

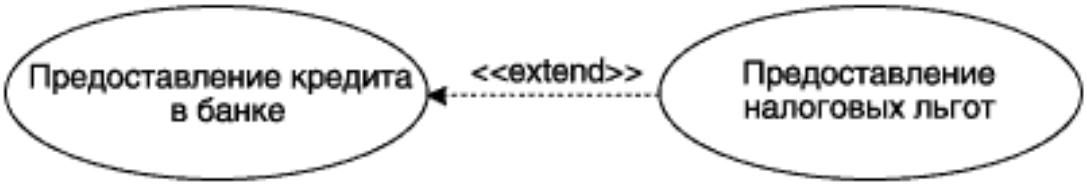


Рисунок 2.1.2 - Пример отношения расширения

Отношение расширения отмечает тот факт, что один из вариантов использования может присоединять к своему поведению некоторое дополнительное поведение, определенное для другого варианта использования. Данное отношение

включает в себя некоторое условие и ссылки на точки расширения в базовом варианте использования. Чтобы расширение имело место, должно быть выполнено определенное условие данного отношения. Ссылки на точки расширения определяют те места в базовом варианте использования, в которые должно быть помещено соответствующее расширение при выполнении условия.

Один вариант использования может быть расширением для нескольких базовых вариантов, а также иметь в качестве собственных расширений несколько других вариантов. Базовый вариант использования может дополнительно никак не зависеть от своих расширений.

Отношение обобщения.

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

При этом дочерние варианты использования участвуют во всех отношениях родительских вариантов.

В свою очередь, дочерние варианты могут наделяться новыми свойствами поведения, которые отсутствуют у родительских вариантов использования, а также уточнять или модифицировать наследуемые от них свойства поведения.

В этом случае дочерний вариант А будет являться специализацией родительского варианта В. При этом В называется предком или родителем по отношению А, а вариант А - потомком по отношению к варианту использования В. Потомок наследует все свойства и поведение своего родителя, а также может быть дополнен новыми свойствами и особенностями поведения. Графически данное отношение обозначается сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительский вариант использования. Пример отношения обобщения приведен на рисунке 2.1.3.

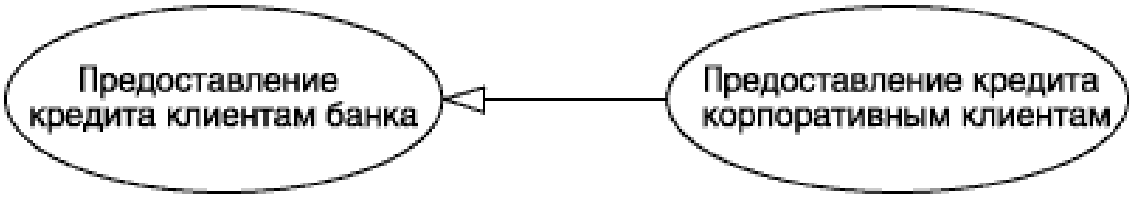


Рисунок 2.1.3 - Пример отношения обобщения

Между отдельными актерами также может существовать отношение обобщения. Данное отношение является направленным и указывает на факт специализации одних актеров относительно других. Например, отношение обобщения от актера А к актеру В отмечает тот факт, что каждый экземпляр актера А является одновременно экземпляром актера В и обладает всеми его свойствами. В этом случае актер В является родителем по отношению к актеру А, а актер А потомком актера В. При этом актер А обладает способностью играть такое же

множество ролей, что и актер B. Графически данное отношение также обозначается стрелкой обобщения.

Отношение включения

Отношение включения - это разновидность отношения зависимости между базовым вариантом использования и его специальным случаем.

Отношение включения между двумя вариантами использования указывает, что некоторое заданное поведение для одного варианта использования включается в качестве составного компонента в последовательность поведения другого варианта использования.

Семантика этого отношения определяется следующим образом. Когда экземпляр первого варианта использования в процессе своего выполнения достигает точки включения в последовательность поведения экземпляра второго варианта использования, экземпляр первого варианта использования выполняет последовательность действий, определяющую поведение экземпляра второго варианта использования, после чего продолжает выполнение действий своего поведения.

При этом предполагается, что даже если экземпляр первого варианта использования может иметь несколько включаемых в себя экземпляров других вариантов, выполняемые ими действия должны закончиться к некоторому моменту, после которого должно быть продолжено выполнение прерванных действий экземпляра первого варианта использования в соответствии с заданным для него поведением.

Один вариант использования может быть включен в несколько других вариантов, а также включать в себя другие варианты. Включаемый вариант использования может быть независимым от базового варианта в том смысле, что он предоставляет ему некоторое инкапсулированное поведение, детали реализации которого скрыты и могут быть перераспределены между несколькими включаемыми вариантами использования. Более того, базовый вариант может зависеть только от результатов выполнения включаемого в него поведения, но не от структуры включаемых в него вариантов.

Отношение включения, направленное от варианта использования A к варианту использования B, указывает, что каждый экземпляр варианта A включает в себя функциональные свойства, заданные для варианта B. Эти свойства специализируют поведение соответствующего варианта A на данной диаграмме. Графически данное отношение обозначается пунктирной линией со стрелкой, которая помечается ключевым словом «include» (включает). Пример отношения включения приведен на рисунке 2.1.4.

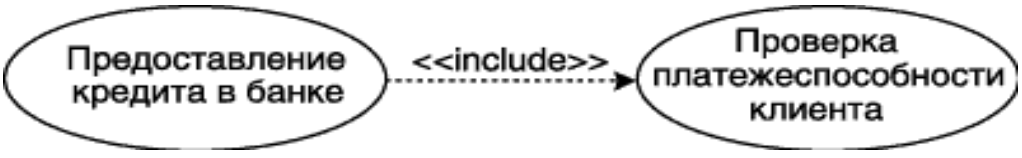


Рисунок 2.1.4 - Пример отношения включения

В качестве примера рассмотрим процесс составления диаграммы вариантов использования для системы Интернет - продажи товаров. В качестве актёров могут выступать два субъекта: продавец, покупатель. Оба они обращаются к сервису «Оформить заказ на покупку товара». Этот сервис выступает в качестве варианта использования разрабатываемой диаграммы, первоначальная структура которой может иметь вид, изображенный на рисунке 2.1.5. Уточненная диаграмма вариантов использования приведена на рисунке 2.1.6.

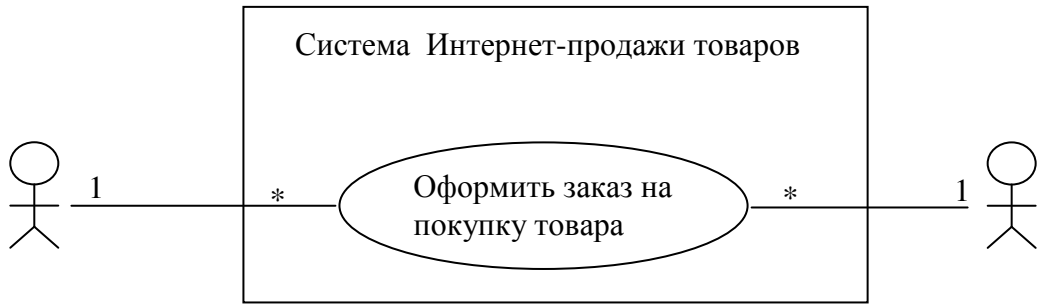


Рисунок 2.1.5 – Исходная диаграмма вариантов использования для примера разработки системы Интернет - продажи товаров

Рисунок 2.1.6 – Уточненный вариант диаграммы

Прпведем также пример диаграммы вариантов использования для программной системы визуализации графовых структур (Рисунок 2.1.7).

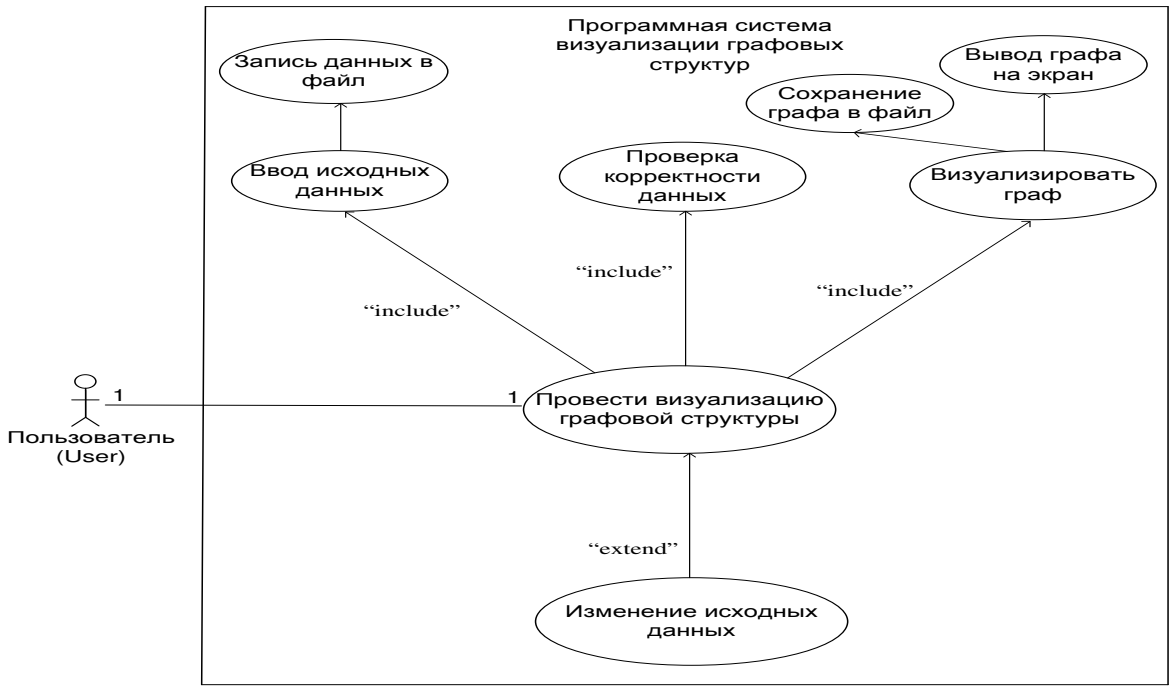


Рисунок 2.1.7 – Диаграмма вариантов использования для программной системы визуализации графовых структур

Приведем рекомендованную последовательность действий при описании вариантов использования и разработке диаграмм вариантов использования.

Последовательность действий:

1. Установить границы системы;
2. Определить главных и второстепенных актеров;
3. Определить цели главных актеров по отношению к системе;
4. Сформулировать основные (базовые) варианты использования, которые специфицируют *функциональные требования* к системе;
5. Изобразить взаимосвязи актеров и базовых вариантов использования с использованием отношения ассоциации;
6. Выделить участников, интересы, предусловия и постусловия выполнения каждого варианта использования;
7. Написать успешный *сценарий* реализации каждого варианта использования (описать основной поток событий);
8. Определить исключения или неуспех в выполнении *сценария* варианта использования (описать альтернативный поток событий);
9. Написать *сценарии* для всех исключений;
10. Выделить дочерние варианты использования и изобразить их взаимосвязи с базовыми с использованием отношений включения и обобщения;
11. Выделить варианты использования для исключений и изобразить их взаимосвязи с базовыми с использованием отношения расширения;
12. Проверить диаграмму на отсутствие дублирования вариантов использования и актеров.

Пример стандартной табличной формы описания варианта использования (п.8, 9 последовательности действий) для процедуры аутентификации пользователя приведен в таблице 2.1.

Таблица 2.1 - Пример табличной формы описания варианта использования

Вариант использования: LogOnRegistrar	
ID: 4	
Краткое описание:	Registrar входит в систему
Главные актеры:	Registrar
Второстепенные актеры:	нет
Предусловия:	1. Registrar еще не вошел в систему
Основной поток: 1.Registrar выбирает LogOn 2.Система запрашивает у Registrar имя пользователя и пароль 3. Registrar вводит имя пользователя и пароль 4. Система принимает имя пользователя и пароль как действительные	
Постусловия:	1. Registrar вошел в систему
Альтернативные потоки: InvalidUserNameAndPassword RegistrarAlreadyLoggedOn	

2.2.2. Диаграммы классов

2.2.2.1 Модель предметной области (диаграмма концептуальных классов)

Модель предметной области (domain model) или *концептуальная объектная модель (conceptual object model)* - представление понятий, атрибутов и ассоциаций из предметной области, имеющих важное значение для решения задачи и выраженных в терминах реального мира.

Объекты – это образующие единое целое элементы, сочетающие в себе данные и функции.

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

Все объекты одного класса должны иметь одинаковый набор операций, одинаковый набор атрибутов и одинаковый набор отношений, но значения атрибутов могут быть различными.

Класс описывает свойства ряда объектов. При этом каждый объект – экземпляр только одного класса.

Графически класс изображается в виде прямоугольника, который может быть разделён горизонтальными линиями на разделы или секции. В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы). Обязательным элементом обозначения класса является его имя. В таком виде классы обозначаются на начальных этапах разработки диаграммы. По мере уточнения отдельных компонентов диаграммы описания классов дополняются атрибутами и операциями. На рисунке 2.1 показаны возможные варианты графического изображения классов.

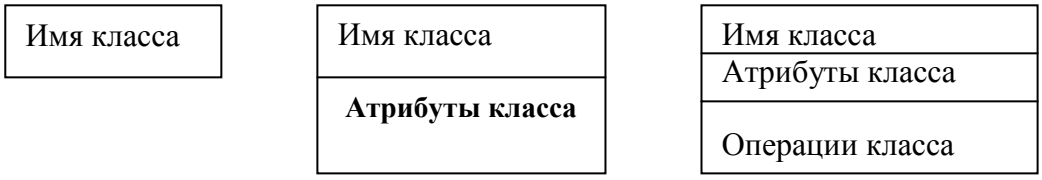


Рисунок 2.2.1 – Возможные варианты графического изображения классов на диаграмме классов

Атрибут (attribute) – содержательная характеристика класса, описывающая множество значений, которые могут принимать отдельные объекты этого класса. Атрибут класса служит для представления отдельного свойства или признака, который является общим для всех объектов данного класса.

В модель предметной области включаются те атрибуты, для которых

- определены варианты использования;
- необходимо хранить определенную информацию (данные).

Типом атрибута может быть другой класс или примитивный тип данных (Boolean, Integer, и т.д.).При моделировании атрибутов применяется принцип инкапсуляции, согласно которому данные, находящиеся внутри объекта в общем случае скрыты и манипулировать ими можно только инициируя одну из функций объекта.

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

Базовые отношения, изображаемые на диаграммах классов, следующие:

1. *Отношение ассоциации (association relationship);*
2. *Отношение агрегации (aggregation relationship);*
3. *Отношение композиции (composition relationship);*
4. *Отношение обобщения (generalization relationship);*
5. *Отношение зависимости (dependency relationship).*

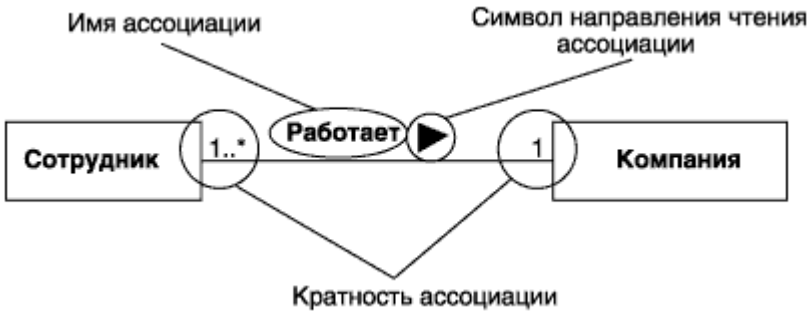
Ассоциация (association) - отношение между классами (экземплярами классов), отражающее значимые связи между ними.

Ассоциация на диаграмме:

– обозначается проведенной между классами линией, с которой связано имя – глагольная форма;

– каждый конец ассоциации называется *ролью (role).*

Ассоциация характеризуется *кратностью*. Кратность (multiplicity) определяет, сколько экземпляров класса А может быть корректно связано с экземпляром класса В в некоторый момент времени. Кратность ассоциации относится к концам ассоциации и обозначается в виде интервала целых чисел («минимум… максимум»). Если кратность не задана явно, значит решение о ней еще не принято. Понятия «кратность по умолчанию» не существует. Пример диаграммы классов приведен на рисунке 2.2.2.



.

Рисунок 2.2.2 – Пример диаграммы классов

Модель на рисунке 2.2.2 с учетом кратностей может быть интерпретирована только единственным образом: «В любой момент времени сотрудник работает в одной и только в одной компании». «В любой момент времени в компании работает не менее одного сотрудника».

Класс может иметь *рефлексивную ассоциацию* – ассоциацию с самим собой. Это означает, что объекты данного класса имеют связь с объектами такого же класса. Пример – каталоги файловой системы (Directory), рисунок 2.2.3.

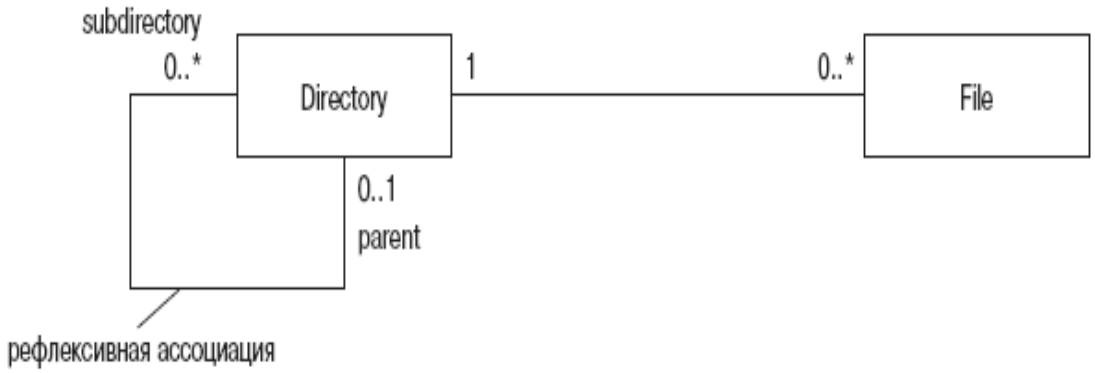


Рисунок 2.2.3 - Рефлексивная ассоциация

Специальной формой или частным случаем отношения ассоциации является *отношение агрегации*, которое, в свою очередь, тоже имеет специальную форму - *отношение композиции*.

Отношение агрегации имеет место между несколькими классами в том случае, если один из классов представляет собой некоторую сущность, включающую в себя в качестве составных частей другие сущности.

Данное отношение имеет фундаментальное значение для описания структуры сложных систем, поскольку применяется для представления системных взаимосвязей типа «часть-целое». Раскрывая внутреннюю структуру системы, отношение агрегации показывает, из каких компонентов состоит система и как они связаны между собой. С точки зрения модели отдельные части системы могут выступать как в виде элементов, так и в виде подсистем, которые, в свою очередь, тоже могут образовывать составные компоненты или подсистемы. Это отношение по своей сути описывает декомпозицию или разбиение сложной системы на более простые составные части, которые также могут быть подвергнуты декомпозиции, если в этом возникнет необходимость в последующем.

Очевидно, что рассматриваемое в таком аспекте деление системы на составные части представляет собой некоторую иерархию ее компонентов, однако данная иерархия принципиально отличается от иерархии, порождаемой отношением обобщения. Отличие заключается в том, что части системы никак не обязаны наследовать ее свойства и поведение, поскольку являются вполне самостоятельными сущностями. Более того, части целого обладают своими собственными атрибутами и операциями, которые существенно отличаются от атрибутов и операций целого.

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой не закрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой «целое». Остальные классы являются его «частями». Пример отношения агрегации приведен на рисунке 2.2.4.

Рисунок 2.2.4 - Отношение агрегации

Отношение композиции является частным случаем отношения агрегации. Это отношение служит для выделения специальной формы отношения «часть-целое», при которой составляющие части в некотором смысле находятся внутри

целого. Специфика взаимосвязи между ними заключается в том, что части не могут выступать в отрыве от целого, то есть с уничтожением целого уничтожаются и все его составные части. Пример отношения агрегации приведен на рисунке 2.2.5.

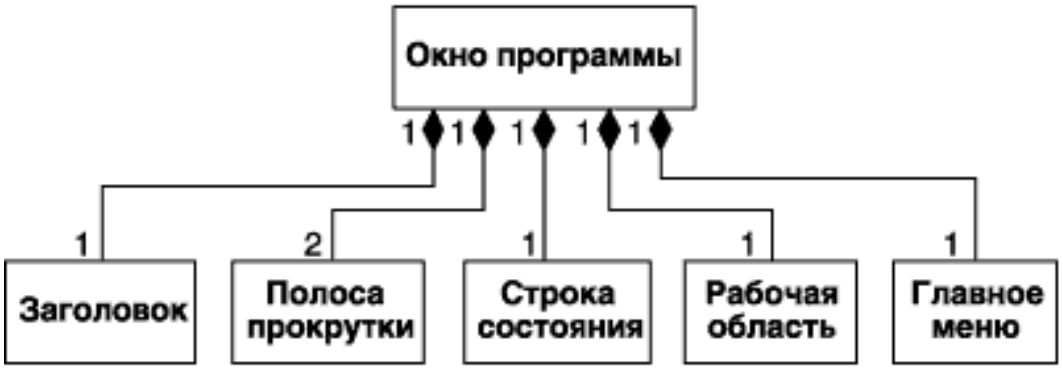


Рисунок 2.2.5 - Отношение композиции

Отношение обобщения является отношением между более общим элементом (предком) и более частным или специальным элементом (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. Стрелка указывает на общий класс (класс-предок или суперкласс), а ее отсутствие - на специальный класс (класс-потомок или подкласс). Пример отношения обобщения приведен на рисунке 2.2.6.

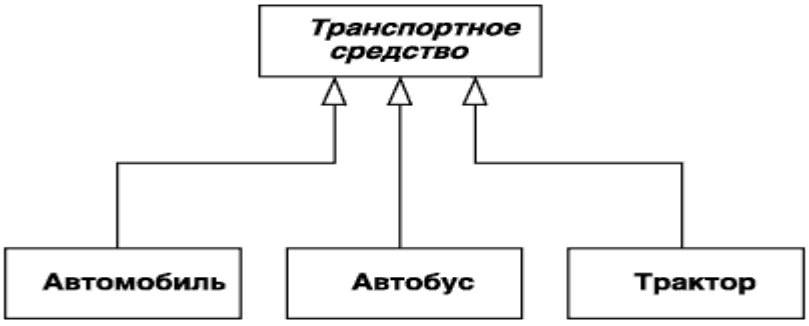


Рисунок 2.2.6. - Отношение обобщения

Пример диаграммы концептуальных классов для программной системы визуализации графовых структур приведен на рисунке 2.2.7.

Рисунок 2.2.7. - Диаграмма концептуальных классов для программной системы визуализации графовых структур

Пример диаграммы концептуальных классов для программной системы построения трехмерных ландшафтов приведен на рисунке 2.2.8.

Рисунок 2.2.8. - Диаграмма концептуальных классов для программной системы построения трехмерных ландшафтов

Для того, чтобы корректно выделить концептуальные классы, следует провести анализ вариантов использования.

Для хорошо спроектированного концептуального класса характерно следующее:

- имя класса отражает его назначение;
- класс является четкой абстракцией, моделирующей один конкретный объект предметной области;
- класс имеет небольшой четко определенный набор функций;
- класс обладает высокой внутренней связностью (все свойства класса служат для реализации его назначения);
- ни один класс не является изолированным.

Выделение концептуальных классов не является тривиальной задачей, для ее решения могут использоваться следующие методы:

1. *Анализ «существительное/глагол».*Анализируется текстовое описание варианта использования. Существительные или именные группы указывают на классы или атрибуты. Глаголы и глагольные группы служат признаком связей или операций.

2. *CRC-анализ* (class-responsibilites-collaborators, класс-обязанности-участники). Основная идея метода состоит в том, что команда разработчиков использует мозговой штурм и заполняет специальные карточки. Вверху такой карточки пишется название класса, в левой половине – за что он отвечает, в правой – с кем сотрудничает. Главным предназначением CRC-карточек является концентрирование на главных абстракциях задачи без учета второстепенных деталей.

3. *Метод «Стереотипов RUP»*. Используется в Rational Unified Process - широко используемом унифицированном итеративном процессе разработки объектно-ориентированных систем. В ходе анализа рассматриваются три разных типа концептуальных классов: «сущность» («entity»), «граница» («boundary») и «управление» («control»). Классы типа «граница» существуют на границе системы и общаются с внешними актерами. Классы типа «управление» координируют поведение системы, соответствующее одному или нескольким вариантам использования. Классы типа «сущность» имеют простое поведение, состоящее в получении и предоставлении информации (например Address – Адрес или Person– Человек). Классы-сущности объединяют несколько вариантов использования, являются объектами, которые контролируют управляющие классы и могут принимать или предоставлять информацию граничным классам. Если в системе есть модель данных, классы-сущности тесно связаны с сущностями или таблицами этой модели.

Модель концептуальных классов должна отображать только стадию анализа проекта. Любые классы, вытекающие только из соображений проектирования (фазы решения) должны быть исключены.

2.2.2.3 Диаграмма классов проектирования

Классы проектирования – это классы, описания которых настолько полны, что они могут быть реализованы.

При моделировании классов проектирования используются следующие источники информации:

- предметная область моделирования. За основу берутся диаграммы концептуальных классов, уточняются, дополняются деталями реализации.
- область решения – техническая и инструментальная база для реализации системы. Для программных систем учитываются особенности языка программирования, существующих библиотек, т.д.

При создании диаграмм классов проектирования используется понятие *квантора видимости* для атрибутов и операций.

Квантор видимости может принимать одно из трех возможных значений и отображается при помощи соответствующих специальных символов:

- «+» обозначает атрибут (операцию) с областью видимости типа общедоступный (public). Атрибут (операция) с этой областью видимости доступен или виден из любого другого класса;
- «#» обозначает атрибут (операцию) с областью видимости типа защищенный (protected). Атрибут (операция) с этой областью видимости недоступен или невиден для всех классов, за исключением подклассов данного класса;
- «-» обозначает атрибут (операцию) с областью видимости типа закрытый (private). Атрибут (операция) с этой областью видимости недоступен или невиден для всех классов без исключения.

Квантор видимости может быть опущен. В этом случае его отсутствие просто означает, что видимость атрибута (операции) не указывается. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

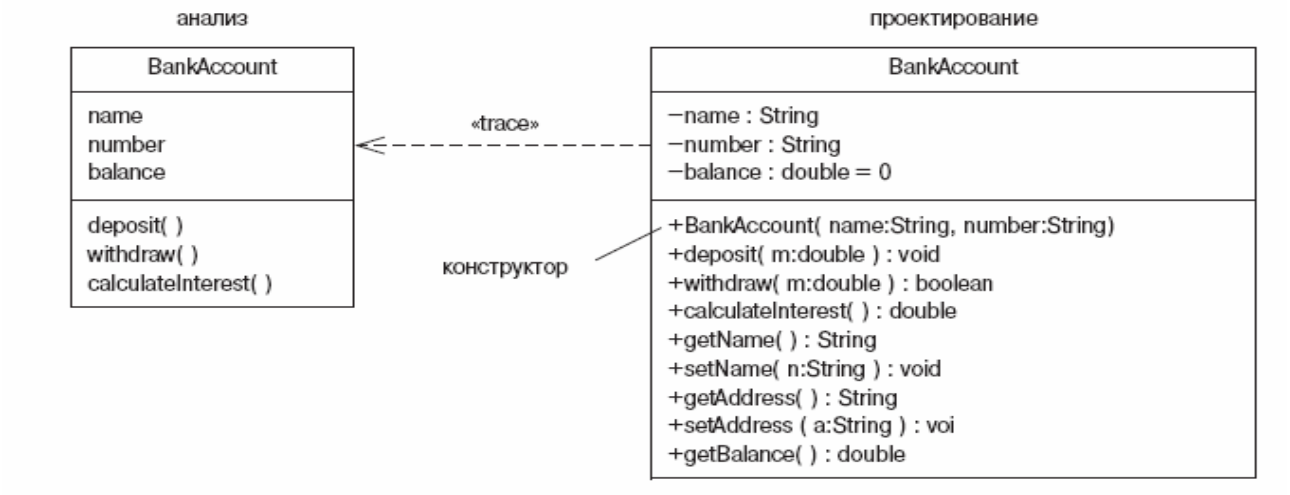


Рисунок 2.2.9 – Детализация обьекта BankAccount на этапе проектирования

Пример фрагмента диаграммы классов проектирования для обьекта BankAccount (БанковскийСчет) приведен на рисунке 2.2.9.

Пример диаграммы классов проектирования для программной системы построения трехмерных ландшафтов приведен на рисунке 2.2.10.

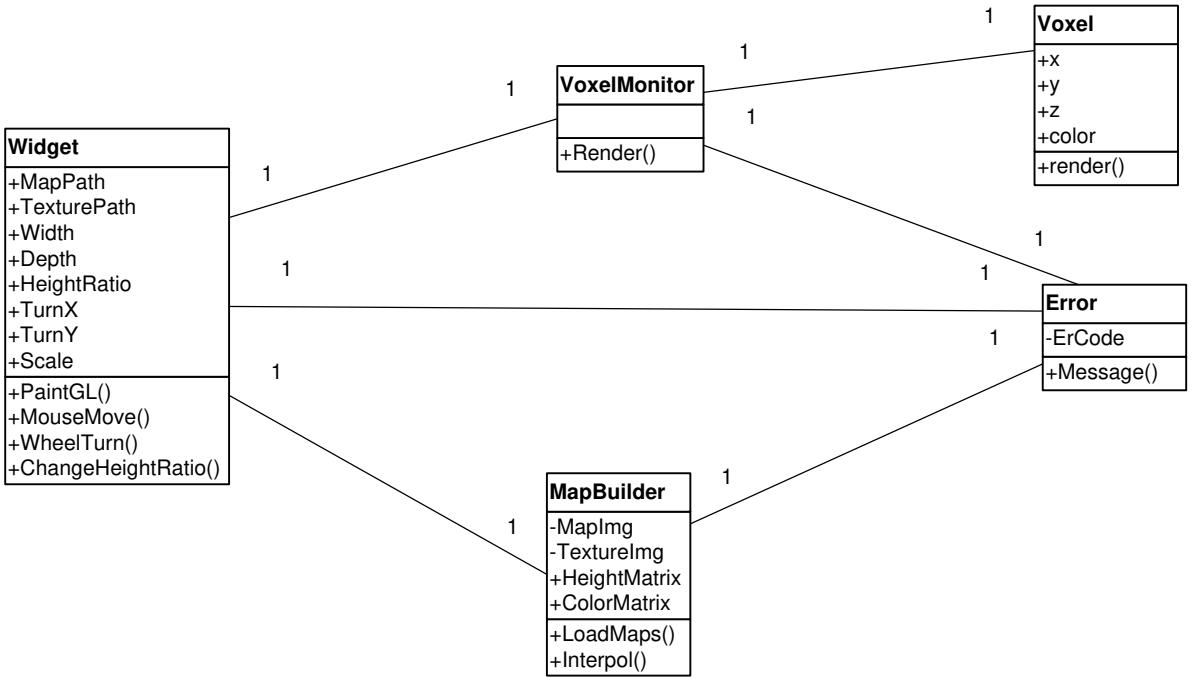


Рисунок 2.2.10. - Диаграмма классов проектирования для программной системы построения трехмерных ландшафтов

Класс проектирования оценивается с точки зрения его пользователей. Критериями хорошо сформулированного класса проектирования являются:

- полнота и достаточность;
- простота;
- высокая внутренняя связность;
- низкая связность с другими классами.

2.2.3. Диаграмма взаимодействия (последовательности)

Диаграмма взаимодействия (interaction diagram) отражает потоки сообщений между объектами системы и вызовы методов – динамическое представление (dynamic view) взаимосвязи объектов. Один из видов - Диаграмма последовательности (sequence diagram) - диаграмма, на которой показаны взаимодействия объектов, упорядоченные по времени их проявления.

На диаграмме последовательности неявно присутствует ось времени, что позволяет визуализировать временные отношения между передаваемыми сообщениями.

Диаграмма последовательности (sequence diagram) - схема, которая для определенного сценария варианта использования показывает:

• генерируемые внешними исполнителями события,
• их порядок
• события, генерируемые внутри системы.

Все системы рассматриваются как «черный ящик».

На диаграмме последовательности изображаются только те объекты, которые непосредственно участвуют во взаимодействии. Ключевым моментом для диаграмм последовательности является динамика взаимодействия объектов во времени. Начальному моменту времени соответствует самая верхняя часть диаграммы.

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый очередностью или степенью активности объектов при взаимодействии друг с другом.

Пример диаграммы последовательности общего вида приведен на рисунке 2.3.1.

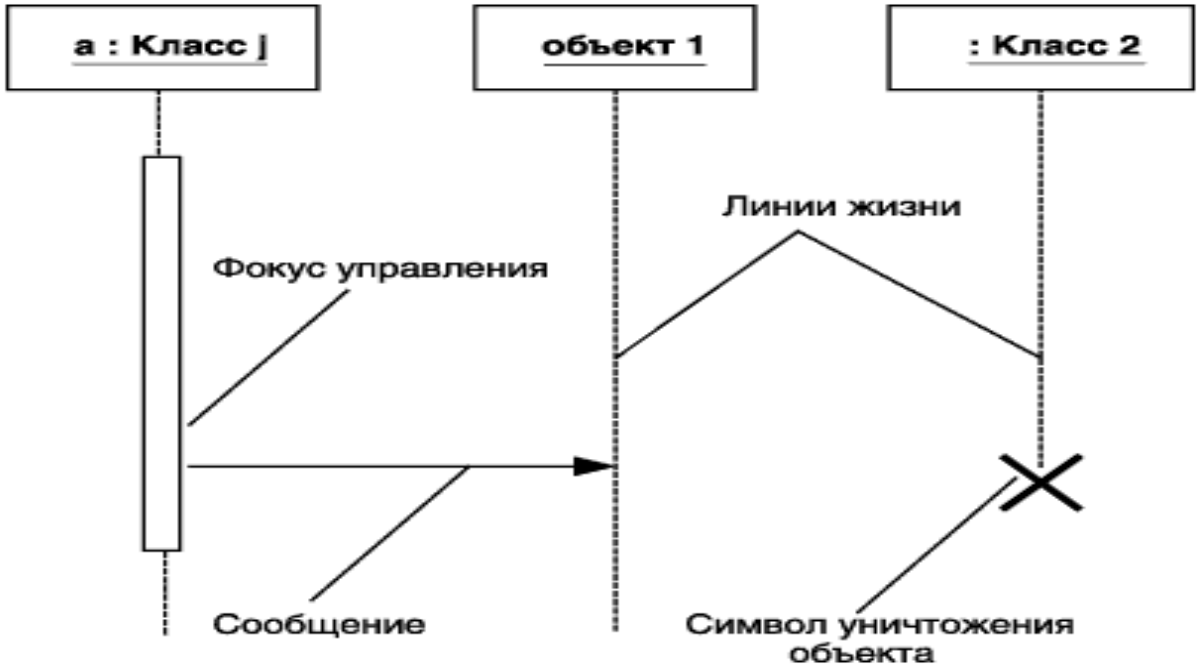


Рисунок 2.3.1. - Диаграмма последовательности

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 плоско-

сти диаграммы последовательности от самой верхней ее части до самой нижней.

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы «X». Ниже этого символа пунктирная линия не изображается, поскольку соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Не обязательно создавать все объекты диаграммы в начальный момент времени. Отдельные объекты могут создаваться по мере необходимости, экономя ресурсы системы и повышая ее производительность. В этом случае прямоугольник объекта изображается не в верхней части диаграммы последовательности, а в той части, которая соответствует моменту создания объекта. При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Объект обязательно создается со своей линией жизни и, возможно, с фокусом управления.

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

Фокус управления изображается в форме вытянутого узкого прямоугольника, верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а его нижняя сторона - окончание фокуса управления (окончание активности). Прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни, если на всем ее протяжении он является активным.

Периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта имеются несколько фокусов управления. Важно сознавать, 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. Чаще всего актер и его фокус управления будут существовать в системе постоянно, отмечая характерную для пользователя активность в инициировании взаимодействий с системой. При этом актер может иметь собственное имя или оставаться анонимным.

Иногда некоторый объект может инициировать рекурсивное взаимодействие с самим собой. 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.

Процесс взаимодействия объектов реализуется посредством *сообщений*, которые посылаются одними объектами другим.

Сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. Прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

Существуют следующие типы сообщений:

- Синхронное сообщение – отправитель ожидает завершения выполнения операции получателем сообщения.
- Асинхронное сообщение – отправитель продолжает исполнение, не ожидая возврата от получателя.
- Возврат – получатель сообщения возвращает фокус управления отправителю сообщения. Могут не указываться на диаграмме.
- Создание объекта.
- Уничтожение объекта.

Сообщения, расположенные на диаграмме последовательности выше, передаются раньше тех, которые расположены ниже. При этом масштаб на оси времени не указывается, поскольку диаграмма последовательности моделирует лишь временную упорядоченность взаимодействий типа «раньше-позже».

Предполагается, что время передачи сообщения достаточно мало по сравнению с процессами выполнения действий объектами, то есть, за время передачи сообщения с соответствующими объектами не может произойти никаких изменений. 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. 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.

В языке UML предусмотрены некоторые стандартные действия, выполняемые в ответ на получение соответствующего сообщения. Эти действия могут быть явно указаны на диаграмме последовательности в форме стереотипа рядом с сообщением, к которому относятся. В этом случае они записываются в кавычках.

Используются следующие *стереотипы сообщений*:

- «call» (вызвать) - сообщение, требующее вызова операции или процедуры принимающего объекта. Если сообщение с этим стереотипом рефлексивное, то оно инициирует локальный вызов операции у самого пославшего это сообщение объекта;

- «return» (возвратить) - сообщение, возвращающее значение выполненной операции или процедуры вызвавшему ее объекту. Значение результата может инициировать ветвление потока управления;
- «create» (создать) - сообщение, требующее создания другого объекта для выполнения определенных действий. Созданный объект может получить фокус управления, а может и не получить его;
- «destroy» (уничтожить) - сообщение с явным требованием уничтожить соответствующий объект. Посылается в том случае, когда необходимо прекратить нежелательные действия со стороны существующего в системе объекта, либо когда объект больше не нужен и должен освободить задействованные им системные ресурсы;
- «send» (послать) - обозначает посылку другому объекту некоторого сигнала, который асинхронно инициируется одним объектом и принимается другим. Отличие сигнала от сообщения заключается в том, что сигнал должен быть явно описан в том классе, объект которого инициирует его передачу.

Кроме стереотипов, 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 ()».

Таким образом, в языке UML каждое сообщение ассоциируется с некоторым действием, которое должно быть выполнено принявшим его объектом. 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.

Для изображения ветвления потока управления рисуются две или более стрелки, выходящие из одной точки фокуса управления объекта. При этом соответствующие условия должны быть явно указаны рядом с каждой из стрелок в форме сторожевого условия. Сторожевые условия должны взаимно исключать одновременную передачу альтернативных сообщений.

В диаграмме последовательности возможно использование *комбинированных фрагментов* (combined fragments), которые разделяют диаграмму на области с различным поведением. Пример на рисунке 2.3.2 иллюстрирует использование операторов opt и alt для формирования комбинированных фрагментов.

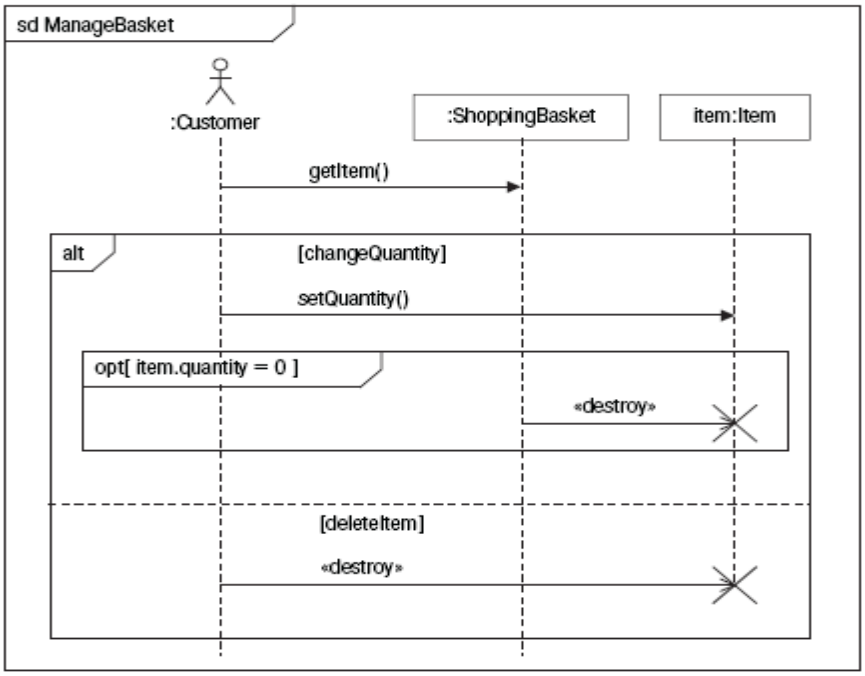


Рисунок 2.3.2. - Диаграмма последовательности варианта использования «Управление корзиной заказов»

Оператор opt создает единственное условие, его действие эквивалентно программному коду

если (условие) тогда
действие.

Оператор alt представляет выбор между рядом альтернатив. Его действие эквивалентно программной конструкции

если (условие1) тогда
действие1
иначе если (условие2) тогда
действие2.

Пример на рисунке 2.3.3 иллюстрирует многократно используемый фрагмент взаимодействия.

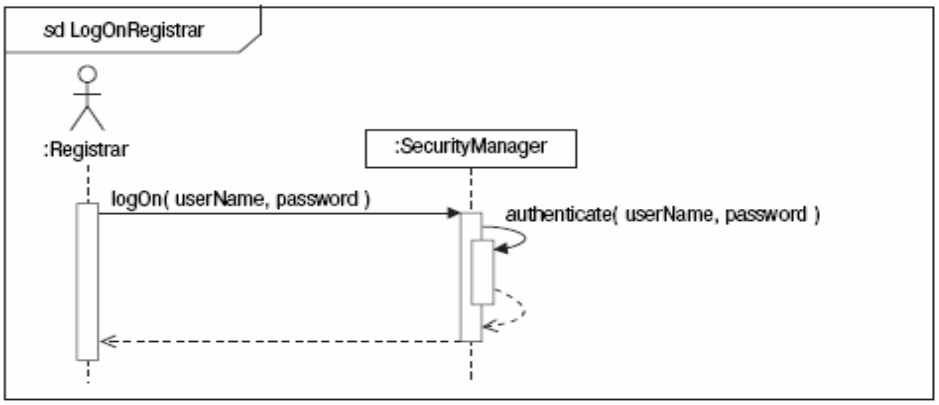


Рисунок 2.3.3. - Диаграмма последовательности варианта использования для процедуры аутентификации пользователя

На рисунках 2.3.4, 2.3.5 приведены примеры диаграмм последовательности.

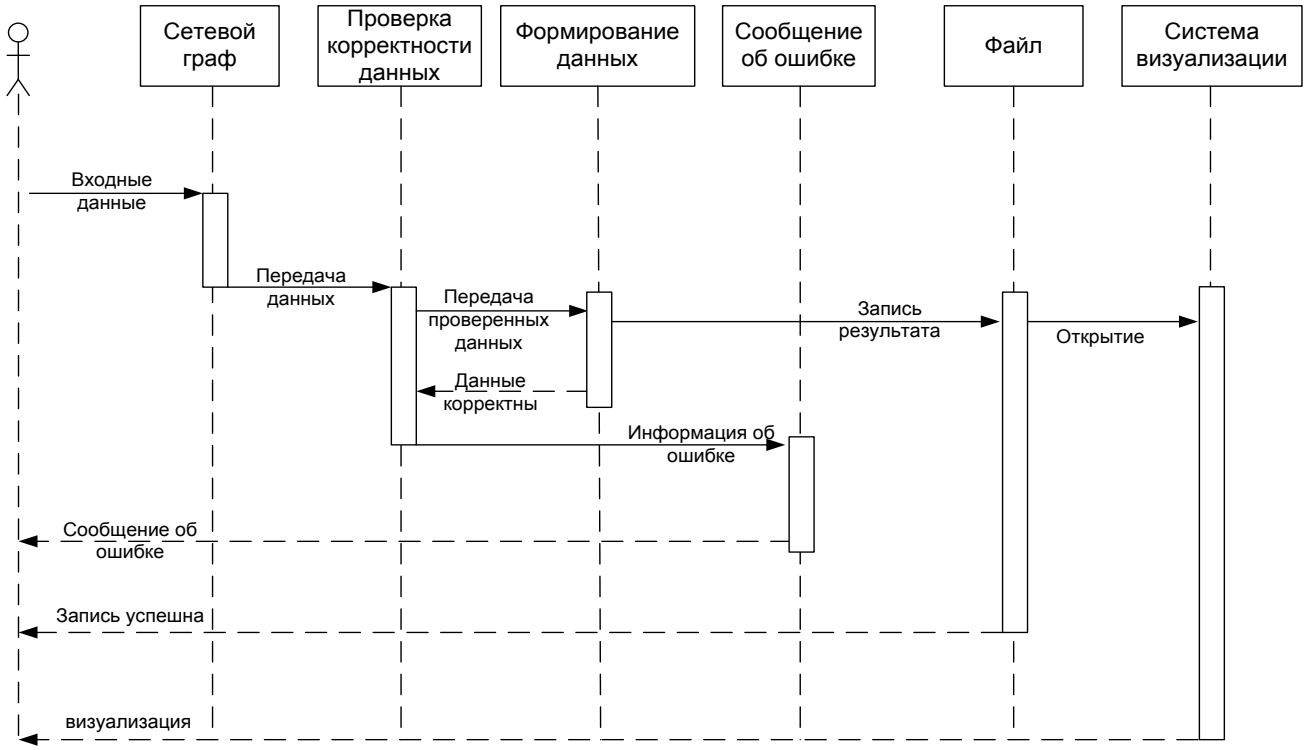


Рисунок 2.3.4 - Диаграмма последовательности
для программной системы визуализации графовых структур

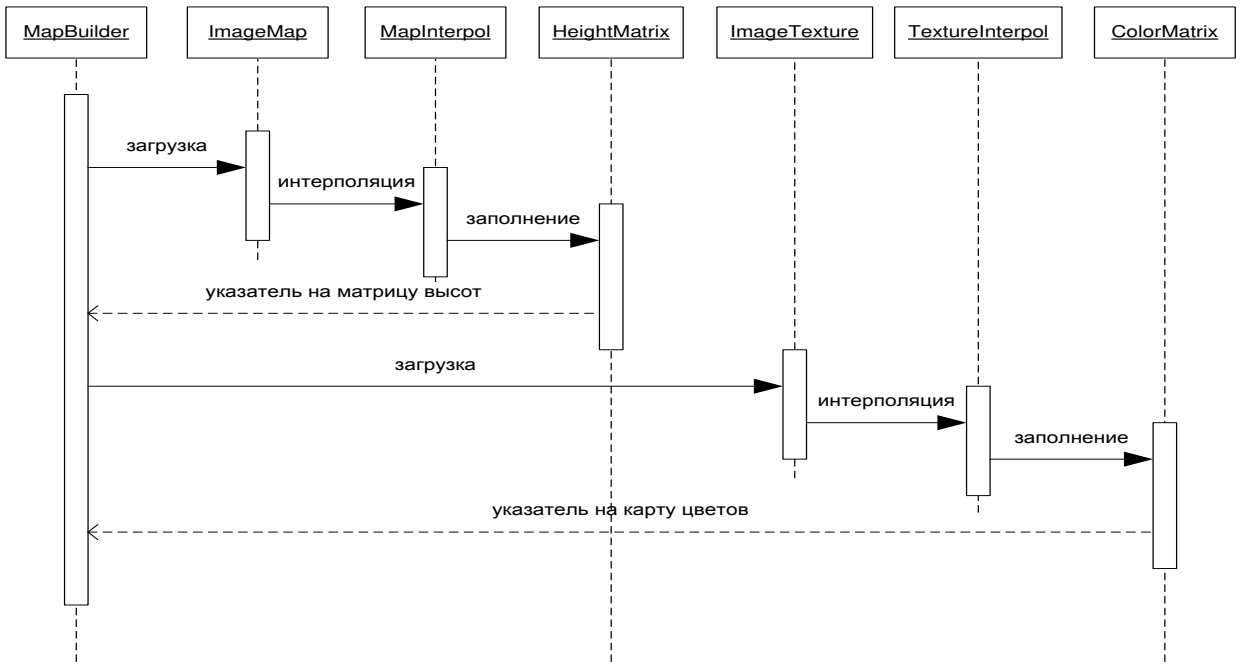


Рисунок 2.3.5 - Диаграмма последовательности
для программной системы построения трехмерных ландшафтов

2.2.4. Диаграмма 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.

Семантика состояния в языке UML имеет ряд специфических особенностей: под *состоянием* понимается абстрактный метакласс, используемый для моделирования отдельной ситуации, в течение которой выполняются некоторые условия. Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта. Изменение отдельных значений атрибутов будет отражать изменение состояния моделируемого класса или объекта.

Диаграмма состояний (state diagram) в UML описывает все возможные состояния объекта и возможные последовательности его переходов из одного состояния в другое, то есть моделирует все изменения состояний объекта как его реакцию на внешние воздействия.

Диаграммы состояний чаще всего используются для описания поведения отдельных объектов, но также могут быть применены для спецификации функциональности других компонентов моделей, таких как варианты использования, актеры, подсистемы, операции и методы.

Диаграмма состояний является графом специального вида, который представляет некоторый *автомат*. Вершинами графа являются возможные состояния автомата, изображаемые соответствующими графическими символами, а дуги обозначают его переходы из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга для более детального представления отдельных элементов модели.

В метамодели UML автомат является пакетом, в котором определено множество понятий, необходимых для представления поведения моделируемой сущности в виде дискретного пространства с конечным числом состояний и переходов.

Длительность нахождения системы в любом из возможных состояний существенно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что в пределе время перехода может быть равно нулю (если дополнительно не оговорено другое), то есть смена состояний объекта может происходить мгновенно.

Поведение автомата моделируется как последовательное перемещение по графу от вершины к вершине с учетом ориентации связывающих их дуг.

Для автомата должны выполняться следующие обязательные условия:

– Состояние, в которое может перейти объект, определяется только его текущим состоянием и не зависит от предыстории;
– В каждый момент времени автомат может находиться только в одном из своих состояний. При этом, автомат может находиться в отдельном состоянии как угодно долго, если не происходит никаких событий;
– Время нахождения автомата в том или ином состоянии, а также время достижения того или иного состояния никак не специфицируются;

- Количество состояний автомата должно быть конечным и все они должны быть специфицированы явным образом. Отдельные псевдосостояния могут не иметь спецификаций (начальное и конечное состояния). В этом случае их назначение и семантика полностью определяются из контекста модели и рассматриваемой диаграммы состояний;
- Граф автомата не должен содержать изолированных состояний и переходов. Для каждого состояния, кроме начального, должно быть определено предшествующее состояние, а каждый переход должен соединять два состояния автомата;
- Автомат не должен содержать конфликтующих переходов, когда объект одновременно может перейти в два и более последующих состояния (кроме случая параллельных подавтоматов). В языке UML исключение конфликтов возможно на основе введения сторожевых условий.

Диаграмма состояний включает следующие элементы:

- состояние (state)**;**
- начальное состояние (start state);
- конечное состояние (end state);
- переход (transition).

Состояние на диаграмме изображается *прямоугольником со скругленными вершинами*. Прямоугольник может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния. При наличии двух секций, в первой из них записывается имя состояния, а во второй список некоторых внутренних действий или переходов в данном состоянии. 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 его содержательный смысл. 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 и этом случае состояние является анонимным. Если на диаграмме анонимных состояний несколько, то они должны различаться между собой.

Список внутренних действий содержит перечень действий или деятельностей, которые выполняются во время нахождения моделируемого элемента в данном состоянии.

Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояние). В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка, из которого может только выходить стрелка, соответствующая переходу.

Конечное состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосостояние). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Оно служит для указания на диаграмме 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, которую может только входить стрелка, соответствующая переходу.

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

Последовательные подсостояния (sequential substates) используются для моделирования такого поведения объекта, во время которого в каждый момент времени объект может находиться в одном и только одном подсостояний. Поведение объекта в этом случае представляет собой последовательную смену подсостояний, начиная от начального и заканчивая конечным. Введение в рассмотрение последовательных подсостояний позволяет учесть более тонкие логические аспекты внутреннего поведения объекта.

Составное состояние может содержать два или более параллельных подсостояний или несколько последовательных подсостояний. Каждое составное состояние может уточняться только одним из указанных способов. При этом любое из подсостояний также может являться составным состоянием и содержать внутри себя другие вложенные подсостояния. Количество уровней вложенности составных состояний не ограничено в языке UML.

Составное состояние может содержать в качестве вложенных подсостояний начальное и конечное состояния. При этом начальное подсостояние является исходным, когда происходит переход объекта в данное составное состояние. Если составное состояние содержит внутри себя конечное подсостояние, то переход в это вложенное конечное состояние означает завершение нахождения объекта в данном вложенном состоянии. Для последовательных подсостояний начальное и конечное состояния должны быть единственными в каждом составном состоянии.

Параллельные подсостояния (concurrent substates) позволяют специфицировать два и более подавтомата, которые могут выполняться параллельно внутри составного события. Каждый из подавтоматов занимает некоторую область или регион внутри составного состояния, которая отделяется от остальных горизонтальной пунктирной линией. Если на диаграмме состояний имеется составное состояние с вложенными параллельными подсостояниями, то объект может одновременно находиться в каждом из этих подсостояний.

Простой переход (simple transition) представляет собой отношение между двумя последовательными состояниями, которое указывает на факт смены одного состояния объекта другим. Если пребывание моделируемого объекта в первом состоянии сопровождается выполнением некоторых действий, то переход во второе состояние будет возможен только после завершения этих действий и, возможно, после выполнения некоторых дополнительных условий, называемых сторожевыми условиями.

В отдельных случаях переход может иметь несколько состояний-источников и несколько целевых состояний. Такой переход получил название *параллельный переход*.

Графически такой переход изображается вертикальной черточкой. Если параллельный переход имеет две и более входящие дуги, его называют *соединением* (join). Если же он имеет две или более исходящие дуги, то его называют *ветвлением* (fork). Текстовая строка спецификации параллельного перехода записывается рядом с черточкой и относится ко всем входящим или исходящим дугам.

Срабатывание параллельного перехода происходит следующим образом. Переход-соединение выполняется, если имеет место событие-триггер для всех исходных состояний этого перехода, и выполнено, если таковое есть, сторожевое условие. При срабатывании перехода-соединения одновременно покидаются все исходные состояния перехода и происходит переход в целевое состояние. При этом каждое из исходных состояний перехода должно принадлежать отдельному подавтомату, входящему в состав автомата.

В случае ветвления происходит разделение автомата на два подавтомата, образующих параллельные ветви вложенных подсостояний. После срабатывания перехода моделируемый объект одновременно будет находиться во всех целевых состояниях этого перехода. Далее процесс изменения состояний будет протекать согласно правилам для составных состояний.

Переход, стрелка которого соединена с границей некоторого составного состояния, обозначает переход в составное состояние. Такой переход эквивалентен переходу в начальное состояние каждого из подавтоматов, входящих в состав данного суперсостояния. Переход, выходящий из составного состояния, относится к каждому из вложенных подсостояний. Это означает, что объект может покинуть составное суперсостояние, находясь в любом из его подсостояний. Для этого вполне достаточно выполнения события и сторожевого условия.

Иногда желательно реализовать ситуацию, когда выход из отдельного вложенного состояния соответствовал бы выходу и из составного состояния тоже. В этом случае изображают переход, который непосредственно выходит из вложенного подсостояния за границу суперсостояния. Аналогично, допускается изображение переходов, входящих извне составного состояния в отдельное вложенное состояние.

На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние.

Каждый переход может быть помечен строкой текста, которая имеет следующий общий формат:

<сигнатура события>'['<сторожевое условие>']' <выражение действия>.

При этом *сигнатура события* описывает некоторое событие с необходимыми аргументами:

<имя события>'('<список параметров, разделенных запятыми>')'.

Событие (event) 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 уже нельзя вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

Семантика понятия события фиксирует внимание на внешних проявлениях качественных изменений, происходящих при переходе моделируемого объекта из состояния в состояние. 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, то есть 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 следовать, после окончания какой деятельности он выполняется. После имени события могут следовать круглые скобки для явного задания параметров соответствующего события-триггера. Если таких параметров нет, то список параметров со скобками может отсутствовать.

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

Введение для перехода сторожевого условия позволяет явно специфицировать семантику его срабатывания. 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.

Выражение действия (action expression) выполняется только при срабатывании перехода. Оно представляет собой атомарную операцию, выполняемую

сразу после срабатывания соответствующего перехода до начала каких бы то ни было действий в целевом состоянии. Атомарность действия означает, что оно не может быть прервано никаким другим действием до тех пор, пока не закончится его выполнение. Данное действие может влиять как на сам объект, так и на его окружение, если это с очевидностью следует из контекста модели.

На рисунке 2.4.1 приведен пример диаграммы состояний для моделирования почтовой программы – клиента, на рисунке 2.4.2 – пример диаграммы состояний для моделирования процесса строительства дома, на рисунке 2.4.3 приведена диаграмма состояний программной системы визуализации графовых структур.

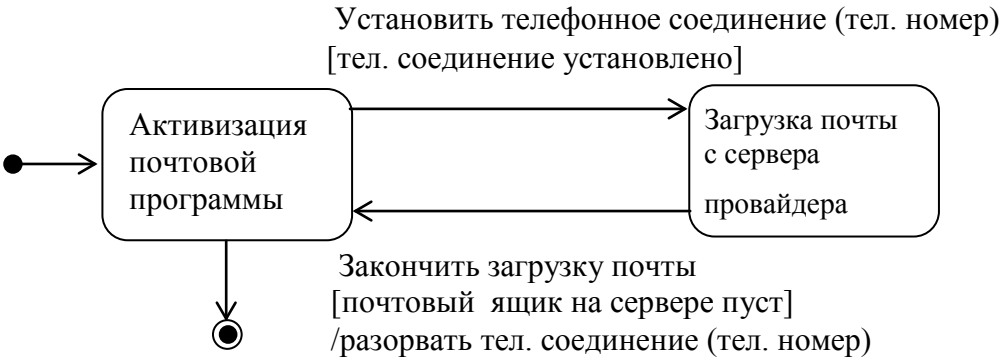


Рисунок 2.4.1 – Диаграмма состояний для моделирования почтовой программы - клиента

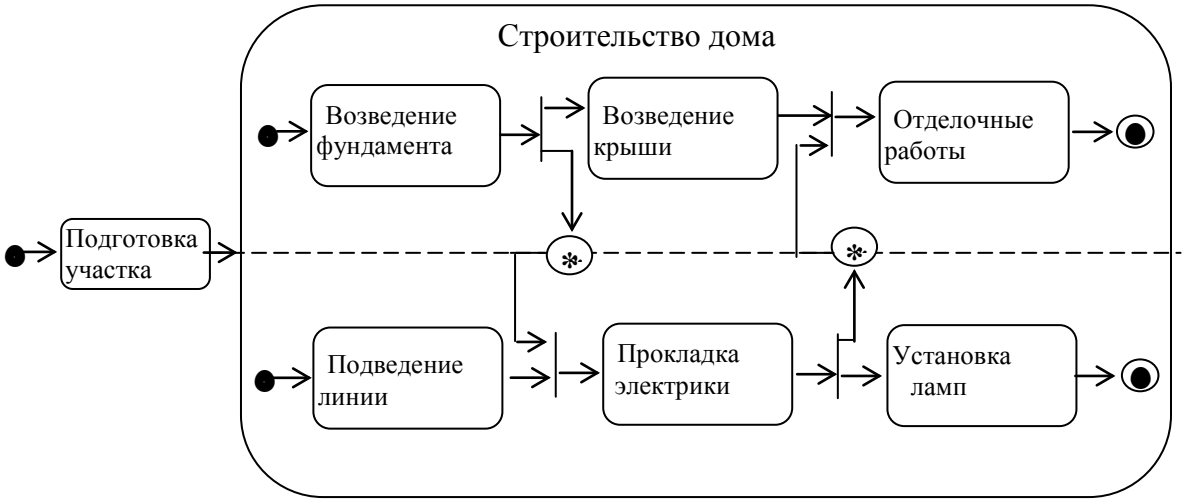


Рисунок 2.4.2 – Диаграмма состояний с использованием составного состояния, параллельных подсостояний, синхронизующих состояний.

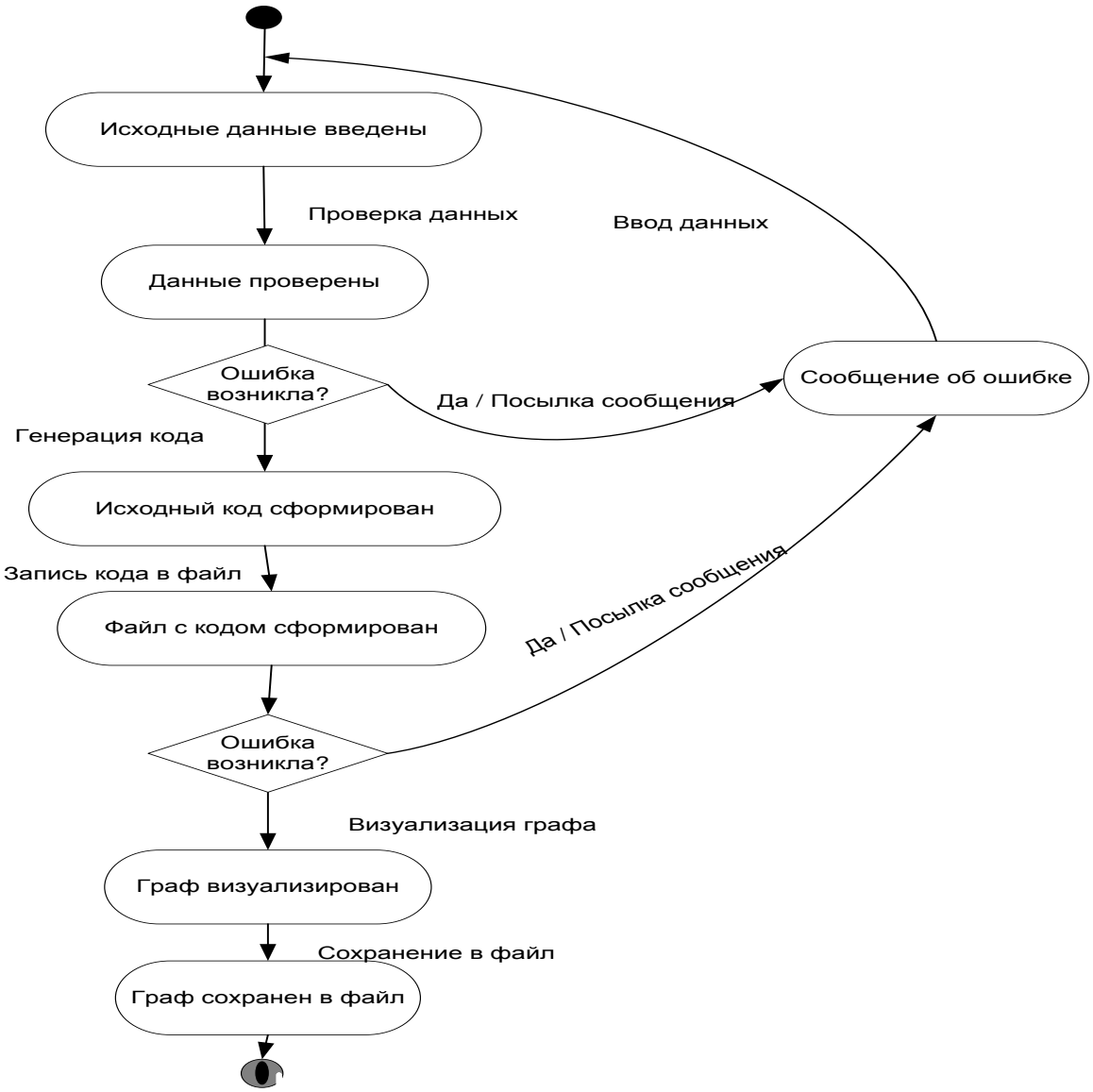


Рисунок 2.4.3 – Диаграмма состояний программной системы визуализации графовых структур.

Рекомендации по построению диаграмм состояний.

По своему назначению диаграмма состояний используется для описания поведения элементов модели с нетривиальным поведением в течение жизненного цикла. При выделении состояний и переходов следует помнить, что длительность срабатывания отдельных переходов должна быть существенно меньшей, чем нахождение моделируемого объекта в соответствующих состояниях. Все переходы должны быть явно специфицированы.

Следует обязательно проверять переходы на отсутствие конфликтности. При наличии параллельности следует заменить конфликтующие переходы одним параллельным переходом типа ветвления.

2.2.5. Диаграмма видов деятельности

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

Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности (диаграммы видов деятельности). Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на этих диаграммах также присутствуют обозначения состояний и переходов. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние выполняется только при завершении этой операции.

Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Они позволяют реализовать в языке UML особенности процедурного и синхронного управления, обусловленного завершением внутренних деятельностей и действий. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации некоторых методов, когда необходимо представить алгоритмы их выполнения.

В контексте языка UML *деятельность* (activity) представляет собой совокупность отдельных вычислений, выполняемых автоматом, приводящих к некоторому результату или действию (action). На диаграмме деятельности отображается логика и последовательность переходов от одной деятельности к другой, а внимание аналитика фокусируется на результатах. Результат деятельности может привести к изменению состояния системы или возвращению некоторого значения.

Диаграмма видов деятельности используется для:

- моделирования бизнес-процессов;
- моделирования вариантов использования;
- моделирования потока между различными вариантами использования;
- моделирования последовательностей выполнения задач;
- моделирования потоков данных и сложных алгоритмов.

Диаграмма видов деятельности включает *узлы (nodes),* соединенные *ребрами (edges).*

Существуют три категории узлов:

1. *Узлы действия (action nodes)* – представляют собой отдельные единицы работы, элементарные с точки зрения рассматриваемой деятельности.
2. *Узлы управления (control nodes)* – управляют потоком деятельности.
3. *Объектные узлы (object nodes)*- представляют объекты, используемые в деятельности.

Ребра представляют потоки деятельности. Существуют два типа ребер:

1. *Ребра потоков управления (control flows).*
2. *Ребра потоков объектов (object flows).*

Основной тип узлов, относящихся *к узлам действия* – это узел вызова действия, или *узел деятельности (activity),* обозначается прямоугольником с закругленными сторонами.

Элемент «деятельность» *(activity)* используется собственно для описания определенной деятельности субъекта или объекта. С этим элементом должно быть связано наименование. Наименование должно отражать цель деятельности. Деятельность именуется глаголом в настоящем времени. На диаграммах деятельности *(activity diagram)* элементы с одним и тем же именем используются для обозначения одного и того же вида деятельности.

Узел деятельности может инициировать деятельность, поведение или операцию. С узлом деятельности могут быть связаны определенные действия, которые происходят на входе этого элемента, на выходе, внутри него или при наступлении определенного события. Действия можно добавить к узлу деятельности при использовании спецификации. Действие может быть описано в форме свободного текста.

Список внутренних действий содержит перечень действий или деятельностей, которые выполняются во время нахождения моделируемого элемента в данном состоянии. Каждое из действий записывается в виде отдельной строки и имеет следующий формат:

<метка-дёйствия '/' выражение-действия>

Метка действия указывает на обстоятельства или условия, при которых будет выполняться деятельность, определенная выражением действия. Перечень меток действия имеет фиксированные значения, которые не могут быть использованы в качестве имен событий. Эти значения следующие:

- entry - указывает на действие, которое выполняется в момент входа в данное состояние (входное действие);
- exit - указывает на действие, которое выполняется в момент выхода из данного состояния (выходное действие);
- do - специфицирует выполняющуюся деятельность («do activity»), которая выполняется в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока не закончится вычисление, специфицированное следующим за ней выражением действия. В этом случае при завершении события формируется соответствующий результат;
- include - используется для обращения к подавтомату, при этом следующее за ней выражение действия содержит имя этого подавтомата.
- event - указывает на действие, которое выполняется в момент наступления определенного события (действие по событию).

Пример узла деятельности с действиями на входе, выходе, внутри узла и по наступлению события представлен на рисунке 2.5.1.

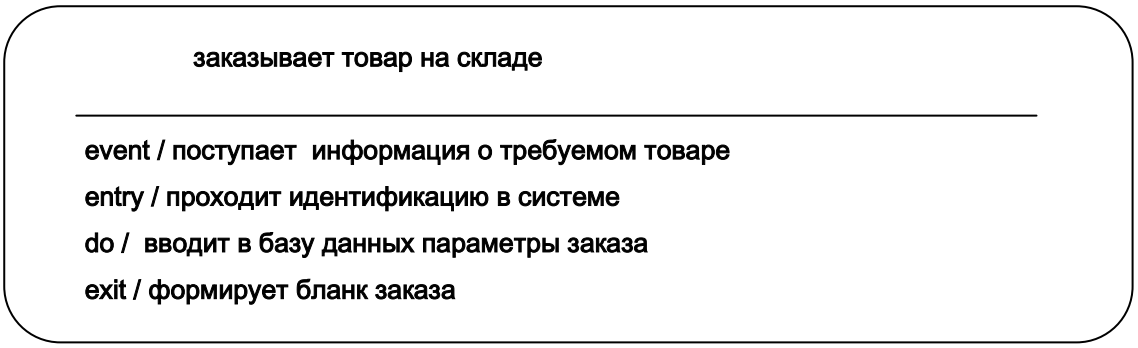


Рисунок 2.5.1 – Пример узла деятельности с перечнем действий.

К *узлам управления* относятся:

- *начальный узел* (initial node);
- *конечный узел* (final node);
- *узел решения* (decision node);
- *узел слияния* (merge node);
- *узел ветвления* (fork node);
- *узел объединения* (join node).

Начальный узел обозначается черным маленьким кружком, с которым может быть связано название, например, «начало».

Конечный узел обозначается большим черным кружком внутри круга, с которым может быть связано название, например, «конец».

Переход между узлами может происходить по условию. Для отображения действий, выполняемых по условию, используется узел решения. На выходе узла решения обязательно сторожевое условие. При соединении потоков управления используется узел слияния. Эти узлы обозначаются в виде ромба. Примеры диаграмм видов деятельности с узлами решения и слияния приведены на рисунках 2.5.2, 2.23.

Узлы ветвления и объединения служат для реализации параллелизма. Примеры приведены на рисунках 2.5.3, 2.5.5, 2.5.6.

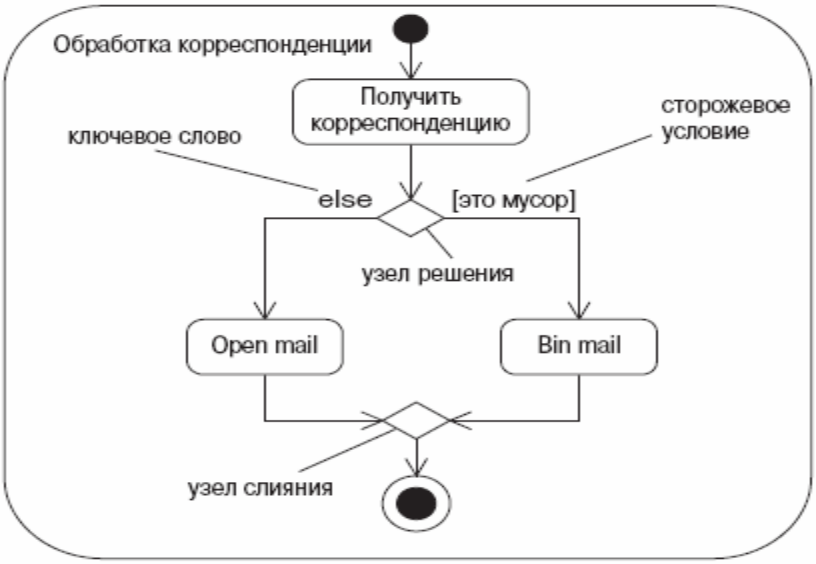


Рисунок 2.5.2 – Диаграмма видов деятельности с узлами решения и слияния.

Объектные узлы на диаграммах деятельности используются для изображения различных сущностей, связанных с определенным видом деятельности. Объект может находиться в определенном состоянии. Объект связывается с деятельностью посредством *потока объектов***.** Поток объектов имеет определенное направление. Примеры приведены на рисунках 2.5.3, 2.5.4.

Чтобы облегчить визуальное восприятие диаграмм деятельности, часто используют разбиение на секции при помощи вертикальных, горизонтальных или кривых *разделительных линий* (*дорожек*, *swimlanes*).

Особенно удобно использовать их при моделировании бизнес – процессов. Действительно, деятельность любой организации представляет собой совокупность отдельных действий, направленных на достижение требуемого результата. 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. Группа состояний между дорожками линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) организации.

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. Примеры приведены на рисунках 2.5.3, 2.5.4.

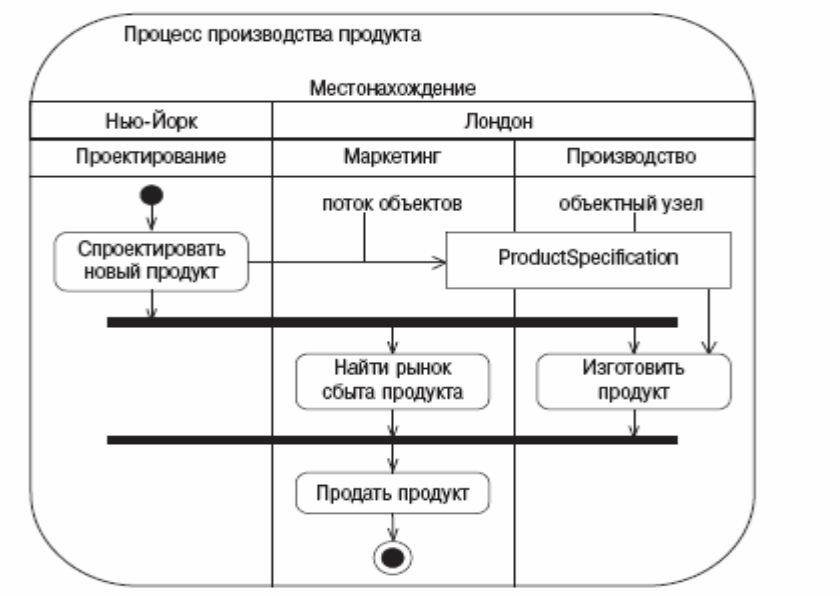


Рисунок 2.5.3 – Диаграмма видов деятельности с узлами ветвления, обьединения и разделительными линиями (дорожками).

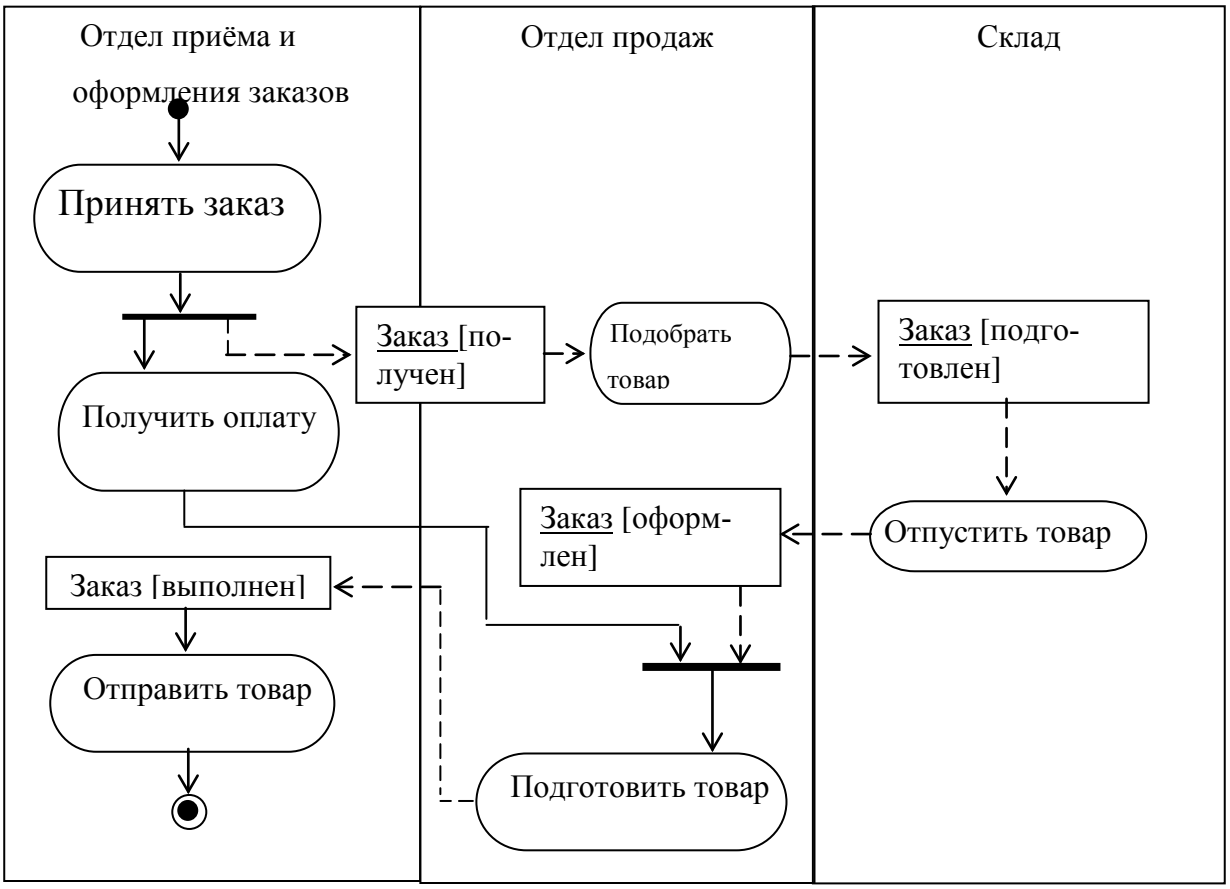


Рисунок 2.5.4 – Диаграмма видов деятельности с указанием потока объектов

Рекомендации по построению диаграмм деятельности.

Диаграмма строится для отдельного класса, варианта использования, отдельной операции класса или целой подсистемы. На начальных этапах проектирования, когда детали реализации деятельностей в проектируемой системе неизвестны, построение диаграммы деятельности начинают с выделения поддеятельностей, которые в совокупности описывают деятельность подсистем. В последующем, по мере разработки диаграмм классов и состояний, эти поддеятельности уточняются в виде отдельных вложенных диаграмм деятельности компонентов подсистем.

При модификации отработанных решений строятся диаграммы деятельности, отражающие конкретные особенности выполнения действий с использованием дорожек и объектов. В последующем такие диаграммы вкладываются в более общие диаграммы деятельности для подсистем и системы в целом.

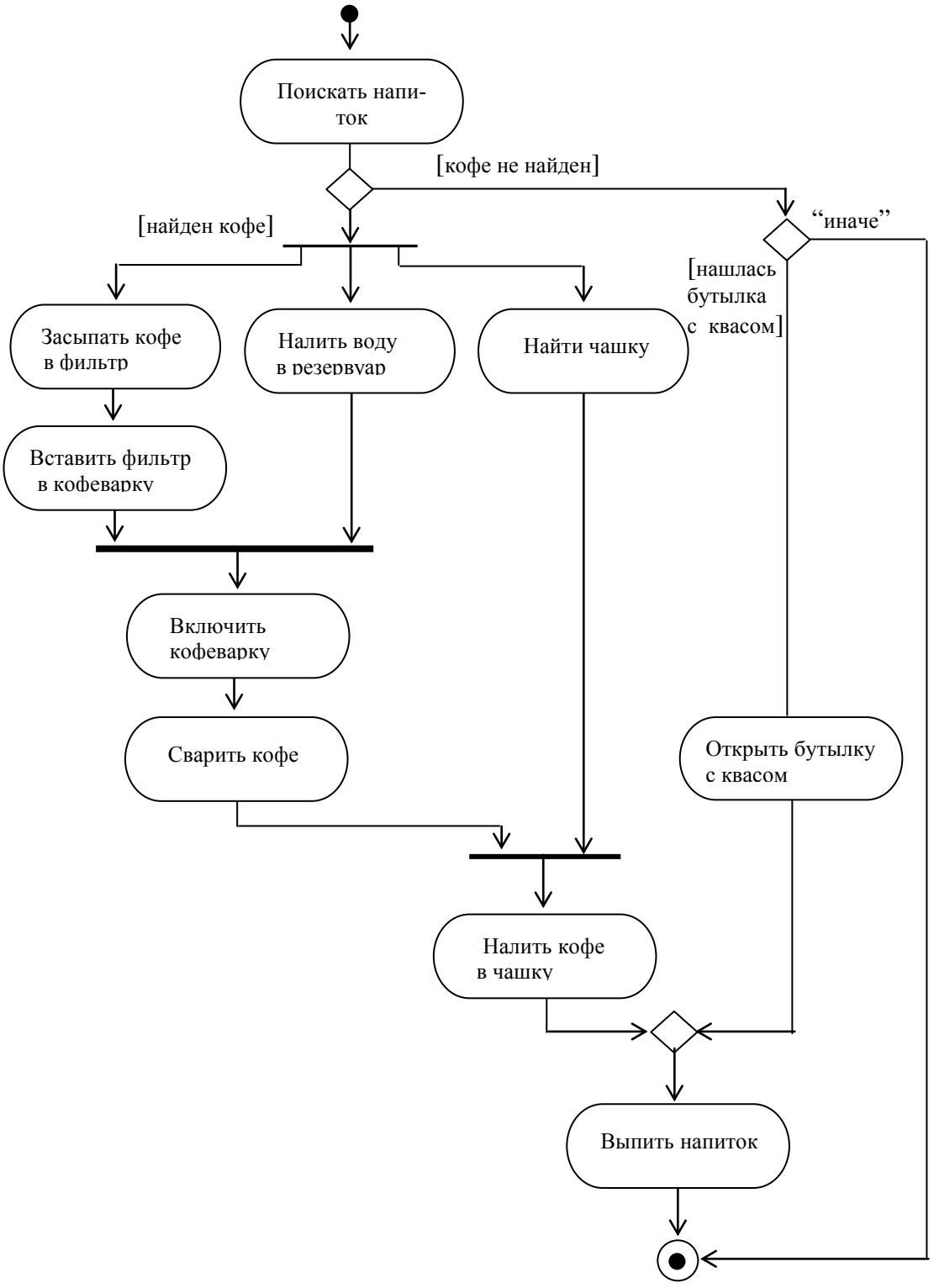


Рисунок 2.5.5 - Диаграмма видов деятельности для примера с приготовлением напитка

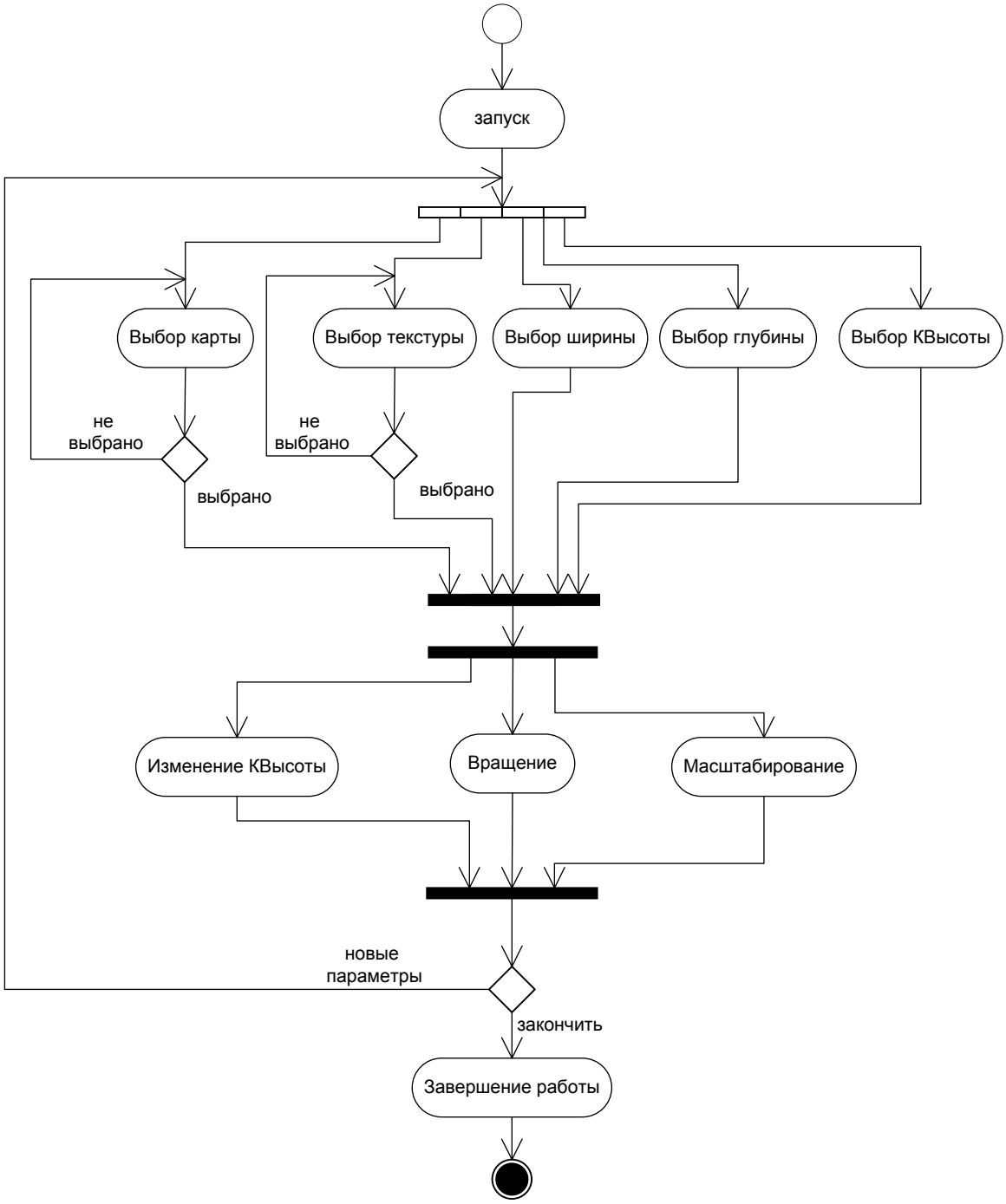


Рисунок 2.5.6 - Диаграмма видов деятельности для программной системы построения трехмерных ландшафтов.

3 КРАТКИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ

Работу над выполнением лабораторной работы следует начать с определения вариантов использования (описания на естественном языке и построения диаграмм).

Затем следует создать модель предметной области (построить и описать диаграмму концептуальных классов).

Следующие этапы проектирования - построение и описание диаграмм взаимодействия, классов проектирования, видов деятельности, состояний. Степень детализации диаграмм выбирает проектировщик.

Более подробные описания и примеры диаграмм UML приведены в [1 - 6].

Разработка диаграмм UML может производиться вручную либо с использованием специализированного программного обеспечения для разработки и графического представления результатов.

ЗАКЛЮЧЕНИЕ

Перспективы дальнейшего развития UML связаны со становлением и интенсивным развитием новой парадигмы объектно-ориентированного анализа - компонентной разработки приложений (Component-Based Development - CBD). Язык UML уже сейчас находит широкое применение в качестве стандарта в процессе разработки программных систем, связанных с такими областями, как моделирование бизнеса, управление требованиями, анализ и проектирование, программирование и тестирование.

Возможно, со временем, на языке UML смогут общаться математики, системные аналитики, физики, программисты, менеджеры, экономисты и специалисты других профессий, представляя свои профессиональные знания в унифицированном виде. Ведь, по существу, каждый из специалистов оперирует модельными представлениями в своей области знаний, и именно этот модельный аспект может быть специфицирован средствами языка UML.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Мартин Фаулер. UML. Основы. / Мартин Фаулер, Кэндалл Скотт.- Спб.- Символ-Плюс, 2002 – 192 с.
2. Крэг Ларман. Применение UML 2.0 и шаблонов проектирования. / Крэг Ларман. - М.: ООО «И.Д. Вильямс»,2007–736 с.
3. Ротко В.Ф. Технологии проектирования программных систем. / Ротко В.Ф., Скатков А.В. - Севастополь, Изд-во СевНТУ, 2006 – 309 с.
4. Описание текущей версии UML www.omg.org
5. Интерактивные спецификации www.uml.org
6. Теория и практика UML www.info-system.ru

Заказ № ___ от «_ __» _____ 2011 г. Тираж ___ экз.
Изд-во СевНТУ