Министерство образования Республики Беларусь

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

Кафедра «Информатика»

Т. А. Трохова

ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ПРАКТИКУМ
по выполнению лабораторных работ для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения

Гомель 2024

УДК 681.3.06(075.8)
ББК 32.973я73
Т76

Рекомендовано научно-методическим советом
факультета автоматизированных и информационных систем
ГГТУ им. П. О. Сухого
(*протокол № 13 от 29.06.2022 г.*)

Рецензент: проф. каф. «Информационные технологии» ГГТУ им. П. О. Сухого
д-р техн. наук, проф. *И. А. Мурашко*

Трохова, Т. А.

Т76 Технологии разработки программного обеспечения : практикум по выполнению лаборатор. работ для студентов специальности 1-40 04 01 «Информатика и технологии программирования» днев. формы обучения / Т. А. Трохова. – Гомель : ГГТУ им. П. О. Сухого, 2024. – 60 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: https://elib.gstu.by. – Загл. с титул. экрана.

Рассмотрены основные возможности проектирования программных комплексов: разработки технического задания на проектирование, применение методов объектно-ориентированного подхода к проектированию, а также структура языка функционального моделирования программных систем UML, приведено описание канонических диаграмм, дано описание подхода к разработке программной системы с их применением. Изложены методические указания к выполнению практических заданий по каждой теме курса.

УДК 681.3.06(075.8)
ББК 32.973я73

Тема 1. Разработка технического задания на проектирование программного комплекса

Теоретические сведения

Процесс разработки программных комплексов и информационных систем имеет несколько стадий, основными из которых являются следующие.

1) Предпроектная стадия.
2) Стадия технического проектирования.
3) Стадия рабочего проектирования.
4) Стадия внедрения и анализа функционирования системы.

Предпроектная стадия включает выполнение следующих работ:

- обследование существующей системы управления и выявление ее недостатков;
- выбор объектов автоматизации;
- предварительный выбор комплекса технических средств (КТС);
- разработка мероприятий по подготовке объектов к внедрению системы;
- составление технического задания на проектирование программного комплекса, в котором обосновывается необходимость разработки системы, дается описание ее структуры, приводится состав КТС, ориентировочные сроки разработки и внедрения и предварительная оценка экономической эффективности.

Стадия технического проектирования предусматривает разработку организационной и функциональной структуры системы, определение уровня автоматизации информационного процесса, определение структуры информационного и технического обеспечения системы и требований к их элементам. На этой стадии проектируются информационно-логические схемы функциональных подсистем программного комплекса. При этом определяются требования к комплексу задач, решение которых необходимо для предоставления информации руководству. Далее выполняется увязка задач, определяются требования к исходным данным. Разрабатывается нормативная база системы. Результаты работы этой стадии оформляются в виде заданий на программирование.

Стадия рабочего проектирования предусматривает выполнение следующих работ:

- составление форм документов и маршрутов их движения;
- разработка технологии ввода, регистрации и корректировки данных;
- выбор типовых процедур обработки данных;
- программирование функциональных задач, их отладка и тестирование;
- комплексная отладка программ в составе подсистем;
- выпуск должностных инструкций;
- организация обучения персонала.

Стадия внедрения и анализа функционирования системы обеспечивает выполнение постепенного перехода от существующей системы управления к разработанной. При внедрении предусматривается подготовка объекта автоматизации, которая заключается в применении новых шифров и форм документов в соответствующих службах, в создании нормативно-справочной информации на компьютерных носителях, в обучении пользователей проверке информации, внесению изменений в данные и решению функциональных задач по инструкции.

После выполнения работ предпроектной стадии последовательно разрабатываются и оформляются следующие документы:

- отчет о предпроектном обследовании;
- технико-экономическое обоснование;
- техническое задание.

Отчет о предпроектном обследовании и результатах анализа материалов обследования содержит следующие разделы.

1. Цель обследования.
2. Основание для обследования.
3. Объект обследования.
4. Организация обследования и состав исполнителей.
5. Программа обследования.
6. Характеристика предприятия.
7. Выводы по результатам анализа организационной структуры предприятия.
8. Результаты обработки опросных листов и других материалов.
9. Выводы по результатам анализа фунциональной структуры.

10. Результаты анализа потоков информации с представлением загрузки информацией отдельных категорий работников, данные об объемах информации и т.д.
11. Выводы по результатам анализа схемы материальных потоков.
12. Выводы по результатам анализа методов планирования и учета.
13. Выводы по результатам анализа уровня автоматизации управленческих работ.
14. Предложения по: совершенствованию организации и функций структуры; совершенствованию потоков информации и форм документов; совершенствованию методов планирования и учета.
15. Выбор и обоснование объектов автоматизации.
16. Обоснование очередности разработки задач.
17. Обоснование предварительного выбора технических средств системы.

Разработка технического задания на создание программного комплекса является завершающим этапом предпроектной стадии и заключается в представлении основных данных для разработки системы, требований к задачам, которые должны быть реализованы автоматизированной системой, а также к техническому комплексу, информационному и математическому обеспечению системы.

Техническое задание (ТЗ) на создание системы должно содержать следующие разделы:

1) общие сведения; 2) назначение и цели создания (развития) системы; 3) характеристика объектов автоматизации; 4) требования к системе; 5) состав и содержание работ по созданию системы; 6) порядок контроля и приемки системы; 7) требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие; 8) требования к документированию; 9) источники разработки.

Ниже приведена краткая характеристика некоторых разделов ТЗ.

Раздел «Требования к системе» состоит из следующих подразделов:

- требования к системе в целом;
- требования к функциям (задачам), выполняемым системой;
- требования к видам обеспечения.

В подразделе «Требования к системе в целом» указывают:

- требования к структуре и функционированию системы;
- требования к численности и квалификации персонала системы и режиму его работы;
- показатели назначения;

- требования к надежности;
- требования безопасности;
- требования к эргономике и технической эстетике;
- требования к транспортабельности для подвижных АС;
- требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы;
- требования к защите информации от несанкционированного доступа и др.

Требования к структуре и функционированию системы следующие:

1) перечень подсистем, их назначение и основные характеристики, требования к числу уровней иерархии и степени централизации системы;

2) требования к способам и средствам связи для информационного обмена между компонентами системы;

3) требования к характеристикам взаимосвязей создаваемой системы со смежными системами, требования к ее совместимости, в том числе указания о способах обмена информацией (автоматически, пересылкой документов, по телефону и т. п.);

4) требования к режимам функционирования системы;

5) требования по диагностированию системы;

6) перспективы развития, модернизации системы.

Требования к надежности представлены следующими:

1) состав и количественные значения показателей надежности для системы в целом или ее подсистем;

2) перечень аварийных ситуаций, по которым должны быть регламентированы требования к надежности, и значения соответствующих показателей;

3) требования к надежности технических средств и программного обеспечения;

4) требования к методам оценки и контроля показателей надежности на разных стадиях создания системы в соответствии с действующими нормативно-техническими документами.

Практическая часть работы

Для всего цикла лабораторных работ необходимо выполнить организационные мероприятия.

Организационные мероприятия

1. Создать фирму по разработке ПО, число сотрудников не более 3 человек.

2. Подписать договор от лица фирмы с заказчиком на разработку ПО (форма любая).

3. Директор закрепляет объекты автоматизации за сотрудниками.

4. Директор разрабатывает план-график выполнения работ (пример в приложении 1).

5. Выполнение этапов работы оформляется отметкой в графе «выполнено» плана графика.

Цель лабораторной работы №1 - получение практических навыков разработки технических заданий на проектирование программных комплексов.

Задание для выполнения практической части работы приведено ниже.

1) Разработать ТЗ на создание программного комплекса для автоматизации выбранной предметной области (организации или предприятия). Организация должна включать несколько структурных подразделений, множество участников процессов, потоки данных и т.д.

2) Документ ТЗ, разработанный и оформленный по приведенному ниже плану, оформить как отчет по лабораторной работе №1. Бланк титульного листа ТЗ приведен в приложении 2.

План выполнения предпроектного обследования и разработки технического задания приведен ниже.

1. Полное и краткое наименование системы
2. Цель создания системы
3. Организационная схема предприятия
4. Процессы. Участники процесса для каждого процесса
5. Требования от участников процесса
6. Функциональные требования для всей системы
7. Разбивка системы на подсистемы по процессам
8. Разбивка подсистем на задачи
9. Выявление функций, входной, выходной и нормативно-справочной информации для каждой задачи

Рассмотрим пример составления технического задания в упрощенной форме для организации «Гомельский государственный технический университет имени П.О. Сухого».

1. Полное и краткое наименование системы

Автоматизированная система «Гомельский государственный технический университет имени П.О. Сухого» (АС ГГТУ)

2. Цель создания системы

АС ГГТУ разрабатывается с целью повышения эффективности управления учебным процессом в различных подразделениях университета.

3. Организационная схема предприятия

Организационная схема

Ректорат
- **Деканаты**
  - **Кафедры**
    - **Заведующий кафедрой**
    - **Преподаватели**
    - **Методист кафедры**
  - **Студенческая группа**
    - **Староста**
    - **Студенты**
  - **Методисты деканата**
  - **Декан**
- **Учебный отдел (УО)**
  - **Отдел расписания**
    - **Диспетчеры отдела расписания**
  - **Методисты УО**

4. Процессы для автоматизации

Из всего многообразия процессов, существующих при работе университета, выделим те, которые будут подлежать автоматизации в разрабатываемом программном комплексе. Каждому процессу соответствует целевая аудитория, т.е. участники процесса. Перечень процессов и участников процессов приведен в табл. 1.1.

5. Требования участников процесса

После проведения анкетирования при предпроектном обследовании предприятия были выявлены те основные требования от участников процессов. Ниже приведен перечень требований от участников процессов.

Таблица 1.1

Перечень процессов и участников процессов

№	Процессы	Участники процессов
1	Мониторинг текущей успеваемости и посещаемости студентов	Студенты, преподаватели, декан, методисты деканата
2	Расчет и мониторинг выполнения учебной нагрузки	Методисты УО, зав. кафедрой, декан, преподаватель, методист кафедры
3	Ведение расписания	Методист кафедры, диспетчер отдела расписания, преподаватель
4	Сессия	Преподаватель, студенты, зав.кафедрой методисты деканата, методист кафедры, декан

Заведующий кафедрой

1) Распределение нагрузки в автоматизированном режиме
2) Получение сведений о результатах аттестации по студентам специальности
3) Получение сведений о выходе студентов на сессию и ее сдаче

Преподаватель

1) Автоматизированная отметка пропусков занятий
2) Ведение индивидуального плана
3) Ведение журнала лабораторных работ
4) Автоматическая простановка и передача в деканат результатов аттестации
5) Получение в электронном виде расписаний дневников, заочников, ИПК

Методист деканата (дневного)

1) Получение от преподавателей оценок аттестации в электронном виде
2) Получать пропуски в электронном виде
3) Выдавать отчеты по аттестации

Методист кафедры

1) Получать от зав. кафедрой нагрузку преподавателей в электронном виде
2) Подводить итоги выполнения нагрузки
3) Формирование и отсылка в УО нарядов на расписание

4) Оформление графиков сдачи зачетов и защиты курсовых

6. Функциональные требования для всей системы

После обработки результатов анкетирования были выделены основные функциональные требования для всего программного комплекса по различным подразделениям. Фрагмент основных требований по процессам «Мониторинг текущей успеваемости и посещаемости студентов» и «Сессия» приведен в табл.1.2.

Таблица 1.2

Основные функциональные требования

Подразделение	Основные требования
Деканат	Вести автоматизированный учет оценок аттестации студентов. Вести нормативно-справочную информацию по студентам. Формировать все формы отчетов по текущей аттестации. Формировать учет и мониторинг посещаемости студентов. Формировать все виды отчетности по посещаемости. Вести автоматизированный учет допусков на сессию и сдачи сессии студентами. Формировать все виды отчетности по результатам сдачи сессии.
Кафедра	Вести индивидуальные планы преподавателей в автоматизированном виде. Вести в автоматизированном виде пропуски занятий студентами. Вести в автоматизированном виде журнала лабораторных работ. Автоматически проставлять и передавать в деканат оценки аттестации студентов. Получать сведения о выходе студентов на сессию и ее сдаче из деканата. Формировать графики сдачи зачетов и защиты курсовых в автоматизированном виде. Вести нормативно-справочную информацию по преподавателям и дисциплинам кафедры.

7. Разбивка системы на подсистемы по процессам

В настоящее время базовым стандартом в области жизненного цикла программных средств и систем является международный стандарт *ISO/IEC 12207: 2008 – Системная и программная инженерия – Процессы жизненного цикла программных средств*. Согласно этого стандарта система состоит из подсистем, которые состоят из задач. Будем придерживаться этой терминологии. Разобъем программный комплекс (систему) на подсистемы согласно автоматизируемым процессам

Автоматизированная система «АС ГГТУ» содержит следующие подсистемы:

- «Текущая успеваемость и посещаемость»;
- «Сессия»;
- «Расписание»;
- «Учебная нагрузка».

8. Разбивка подсистем на задачи

Подсистема «Текущая успеваемость и посещаемость» включает следующие задачи:

- «Текущая успеваемость - деканат»;
- «Посещаемость - деканат»;
- «Текущая успеваемость - преподаватель».

Подсистема «Сессия» включает следующие задачи:

- «Сессия – деканат»;
- «Сессия – преподаватель».

Подсистема «Учебная нагрузка» включает следующие задачи:

- «Учебная нагрузка - кафедра»;
- «Учебная нагрузка - преподаватель».

Подсистема «Расписание» включает следующие задачи:

- «Расписание – кафедра»;
- «Расписание – учебный отдел».

Подсистема «Расчет учебной нагрузки» включает следующие задачи:

- «Учебная нагрузка– кафедра»;
- «Учебная нагрузка– преподаватель».

9. Выявление функций, входной, выходной и нормативно-справочной информации для каждой задачи

«Текущая успеваемость - деканат»

Основные функции:

- ввод оценок аттестации;
- автоматизированное формирование справки об аттестации студента;
- автоматизированное формирование отчета по результатам аттестации группы;
- автоматизированное формирование отчета по результатам аттестации факультета.

Входная информация:

- аттестационные листы групп;

Выходная информация

- таблица БД «Оценки аттестации»
- отчет по итогам аттестации факультета
- отчет о рейтинге студентов

Нормативно-справочная информация

- справочник факультетов;
- справочник групп;
- справочник студентов;
- справочник кафедр;
- справочник преподавателей;
- справочник предметов.

«Посещаемость - деканат»

Основные функции:

- формирование таблицы БД «Пропуски занятий»
- ввод данных по посещаемости группы;
- автоматизированное формирование справки о посещаемости студента;
- автоматизированное формирование отчета по посещаемости группы;
- автоматизированное формирование отчета по посещаемости факультета.

Входная информация:

- лист посещаемости;
- таблица БД плановой нагрузки по группе (из подсистемы «Расчет нагрузки»

Выходная информация

- таблица БД «Пропуски занятий»

- отчет по факультету;
- отчет по группе;
- справка о посещении студентом занятий;
- письмо родителям.

Нормативно-справочная информация

- справочник факультетов;
- справочник групп;
- справочник студентов.

«Текущая успеваемость - преподаватель»

Основные функции:

- формирование таблицы БД «Журнал лабораторных работ»
- заполнение электронного журнала лабораторных работ;
- автоматизированное формирование оценок аттестации;
- автоматизированное формирование отчета преподавателя по результатам аттестации групп;
- просмотр журнала студентами.

Входная информация:

- журнал (бумажный) по лабораторным работам

Выходная информация

- таблица БД «Журнал лабораторных работ»
- таблица БД «Оценки аттестации»;
- отчет «Журнал лабораторных работ» для преподавателей
- отчет «Журнал лабораторных работ» для студентов

Нормативно-справочная информация

- справочник факультетов;
- справочник групп;
- справочник студентов;
- справочник кафедр;
- справочник преподавателей;
- справочник предметов;
- справочник состояний.

В приложении 3 приведено описание задач для других подсистем автоматизированной системы «АС ГГТУ».

Примеры вариантов предметных областей (организаций, предприятий) для разработки технического задания приведены в приложении 4.

Тема 2. Диаграмма вариантов использования (диаграмма прецедентов)

Теоретические сведения
Общая характеристика языка UML

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

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

Основным компонентом языка UML является диаграмма.

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

В нотации языка UML определены следующие виды канонических диаграмм:

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

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

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

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

Диаграмма вариантов использования (диаграмма прецедентов)

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

Создание диаграммы вариантов использования имеет следующие цели:

- определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы;
- сформулировать общие требования к функциональному поведению проектируемой системы;
- разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей;
- подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

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

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

Отдельный вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его краткое имя в форме существительного или глагола с пояснительными словами. Сам текст имени варианта использования должен начинаться с заглавной буквы.

Имя (name) — строка текста, которая используется для идентификации любого элемента модели (см. рис. 2.1).

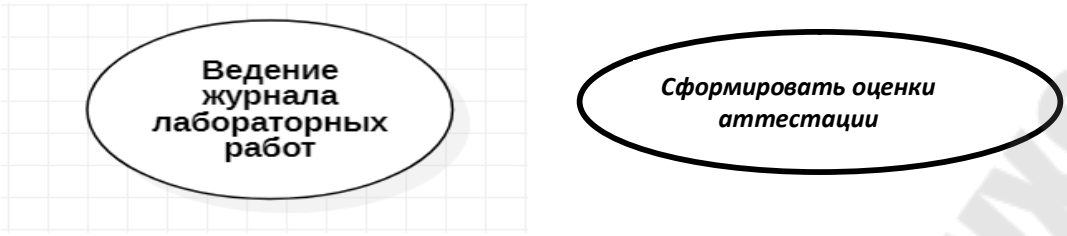


Рис. 2.1. Графическое обозначение вариантов использования

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования.

Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается имя актера (см. рис. 2.2).

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

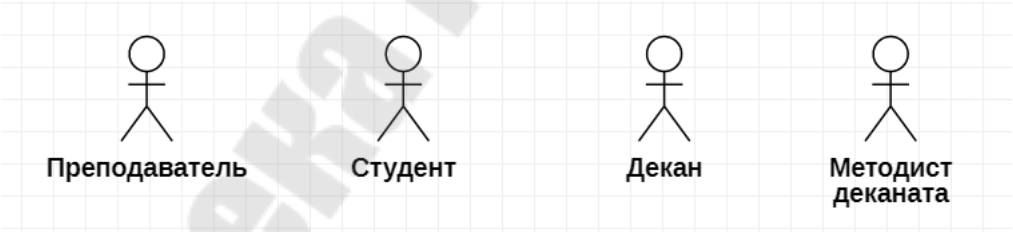


Рис. 2.2. Графическое обозначение актера

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

Между элементами диаграммы вариантов использования могут существовать различные отношения, которые описывают

взаимодействие экземпляров одних актеров и вариантов использования с экземплярами других актеров и вариантов.

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

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

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

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

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

На диаграмме вариантов использования, так же как и на других диаграммах, отношение ассоциации обозначается сплошной линией между актером и вариантом использования (см. рис. 2.3).

Рис. 2.3. Отношение ассоциации

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

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

Отношение включения устанавливается только между двумя вариантами использования и указывает на то, что заданное поведение для одного варианта использования включается в качестве составного фрагмента в последовательность поведения другого варианта использования (рис. 2.4).

Рис.2.4. Отношение включения

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

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

Отношение расширения между вариантами использования обозначается как отношение зависимости в форме пунктирной линии

со стрелкой, направленной от того варианта использования, который является расширением для базового варианта использования.

Данная линия со стрелкой должна быть помечена стереотипом <<extend>> (рис. 2.5).

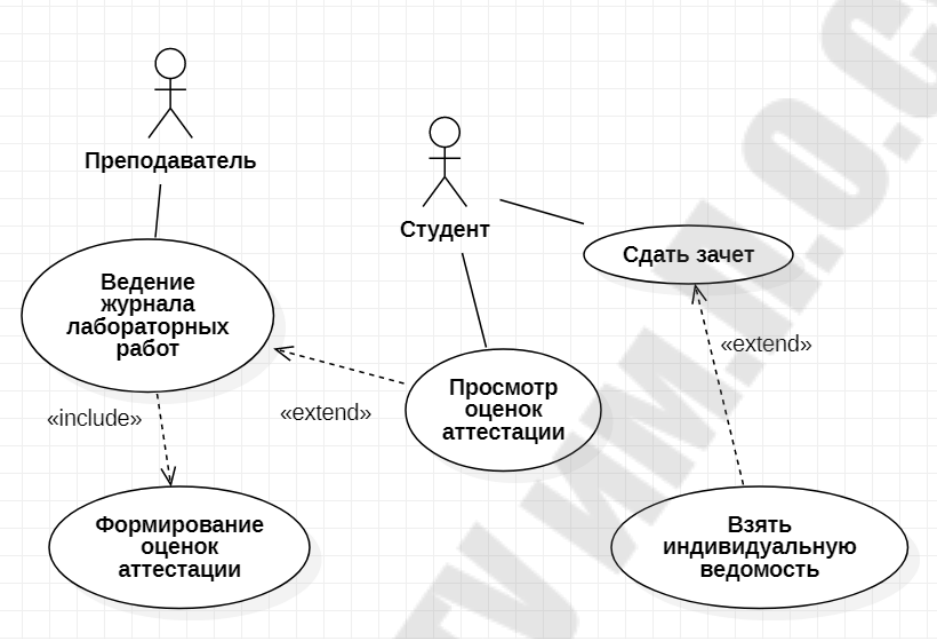


Рис. 2.5. Отношение расширения

Отношение обобщения служит для указания того факта, что некоторый вариант использования A может быть обобщен до варианта использования B, который называется предком или родителем по отношению A, а вариант A — потомком по отношению к варианту использования B. Графически данное отношение обозначается сплошной линией со стрелкой в форме незакрашенного треугольника, которая указывает на родительский вариант использования.

Отношение обобщения между вариантами использования применяется в том случае, когда необходимо отметить, что дочерние варианты использования обладают всеми атрибутами и особенностями поведения родительских вариантов. Следует учитывать, что дочерние варианты использования участвуют во всех отношениях родительских вариантов. Пример применения отношения обобщения приведен на рис. 2.6.

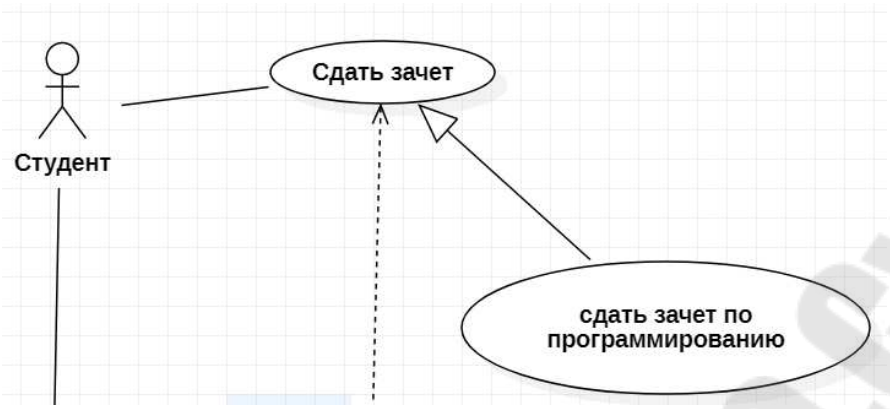


Рис. 2.6. Пример применения отношения обобщения

Пример диаграммы вариантов использования для функциональной модели «Составление расписания» приведен на рис. 2.7.

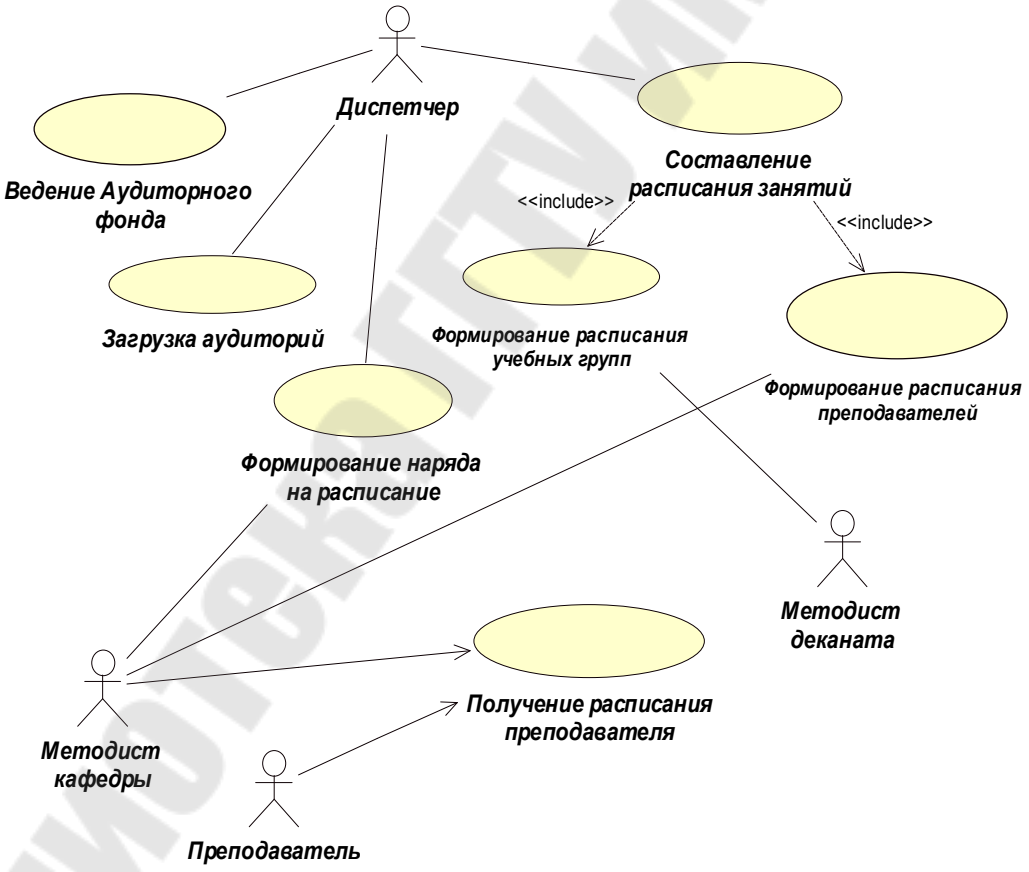


Рис. 2.7. Пример диаграммы вариантов использования

Сценарии использования (потоки событий)

Диаграмма вариантов использования предоставляет знание о необходимой функциональности конечной системы в интуитивно-понятном виде, однако не несет сведений о фактическом способе ее реализации. Конкретные варианты использований могут звучать слишком общно и расплывчато и не являются пригодными для программистов. Для документации вариантов использования в виде некоторой спецификации и для устранения неточностей и недоразумения диаграмм вариантов использований, как часть процесса сбора и анализа требований составляются так называемые **сценарии использования.**

Сценарии использования — это текстовые представления тех процессов, которые происходят при взаимодействии пользователей системы и самой системы. Они являются четко формализованными пошаговыми инструкциями описывающими тот или иной процесс в терминах шагов достижения цели. Сценарии использования однозначно определяют конечный результат. Сценарии использования описывают варианты использования на естественном языке.

Они не имеют общего, шаблонного вида написания, однако рекомендуется следующий вид:

1. Предусловия — условия, которые должны быть выполнены для выполнения данного варианта использования;
2. Постусловия — что получается в результате выполнения данного варианта использования;
3. Взаимодействующие стороны;
4. Краткое описание;
5. Основной ход событий (Главный поток);
6. Под-потоки
7. Альтернативные потоки
8. Исключения;
9. Примечания.

Практическая часть темы 2

Необходимо разработать диаграммы вариантов использования для программной системы по индивидуальному заданию, для этого нужно выполнить следующие действия.

1. Открыть новый проект в StarUML.
2. Определить актеров и варианты использования и присвоить им имена согласно предметной области.
3. Определить виды связи между ними, выделив ассоциативные связи как основные.
4. Определить связи расширения и включения, если это необходимо при описании функций прецедентов.
5. Построить диаграмму или диаграммы прецедентов по разработанному техническому заданию.
6. Присвоить имя диаграмме согласно предметной области и решаемой задаче.
7. Разработать сценарии использования для двух-трех прецедентов.
8. Оформить отчет по лабораторной работе как приложение к техническому заданию.

Тема 3. Диаграмма классов

Краткие теоретические сведения

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

Диаграммы классов создаются при логическом моделировании ПС и служат для следующих целей.

1) Для моделирования данных. Анализ предметной области позволяет выявить основные характерные для нее сущности и связи между ними. Это удобно моделируется с помощью диаграмм классов. Эти диаграммы являются основой для построения концептуальной схемы базы данных.

2) Для представления архитектуры ПС. Можно выделить архитектурно значимые классы и показать их на диаграммах, описывающих архитектуру программной системы.

3) Для моделирования навигации экранов. На таких диаграммах показываются пограничные классы и их логическая

взаимосвязь. Информационные поля моделируются как атрибуты классов, а управляющие кнопки – как операции и отношения.

4) Для моделирования логики программных компонент.

5) Для моделирования логики обработки данных.

Основные компоненты диаграммы классов:

- классы;
- отношения;
- интерфейсы.

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

Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 3.1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

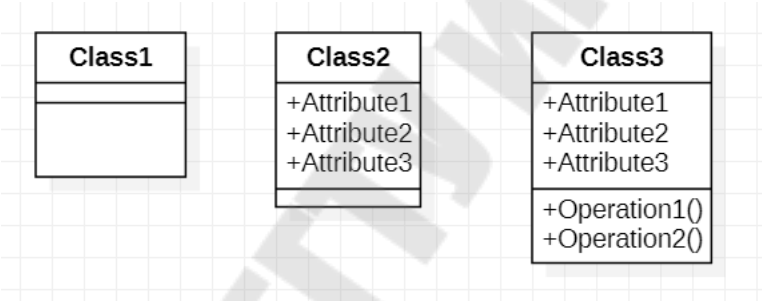


Рис. 3.1. Графическое обозначение класса

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

Во второй сверху секции прямоугольника класса записываются его атрибуты (attributes) или свойства. В языке UML принята определенная стандартизация записи атрибутов класса, которая подчиняется некоторым синтаксическим правилам. Каждому

атрибуту класса соответствует отдельная строка текста, которая состоит из:

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

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

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

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

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

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

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

Имя атрибута является единственным обязательным элементом синтаксического обозначения атрибута.

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

- [0..1] означает, что кратность атрибута может принимать значение 0 или 1. При этом 0 означает отсутствие значения для данного атрибута.
- [0..*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 0. Эта кратность может быть записана короче в виде простого символа — [*].

– [1.:*] означает, что кратность атрибута может принимать любое положительное целое значение большее или равное 1.
– [1..5] означает, что кратность атрибута может принимать любое значение из чисел: 1, 2, 3, 4, 5.

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

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

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

– Символ "+" обозначает операцию с областью видимости типа общедоступный (public).
– Символ "#" обозначает операцию с областью видимости типа защищенный (protected).
– Символ "-" используется для обозначения операции с областью видимости типа закрытый (private).

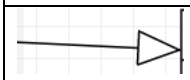
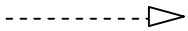
Кроме внутреннего устройства или структуры классов на соответствующей диаграмме указываются различные отношения между классами.

Базовые отношения или связи в языке UML приведены в табл. 3.1.

Таблица 3.1

Базовые отношения

Обозначение	Наименование	Назначение
1	2	3
	отношение зависимости (dependency relationship)	один класс зависит от определений, сделанных в другом
	отношение ассоциации (association relationship)	наличие некоторого отношения между классами

1	2	3
	отношение зависимости (агрегирование)	частный случай ассоциации - отношение типа "часть/целое"
	отношение зависимости (композиция)	специальной формы отношения "часть-целое", при которой составляющие части в некотором смысле находятся внутри целого
	отношение обобщения (generalization relationship)	отношение между более общим элементом (родителем) и более частным элементом (потомком)
	отношение реализации (realization relationship)	Отношение между классом и интерфейсом

Отношение ассоциации соответствует наличию некоторого отношения между классами (рис.3.2). Имя ассоциации является необязательным элементом ее обозначения. Если оно задано, то записывается рядом с линией соответствующей ассоциации.

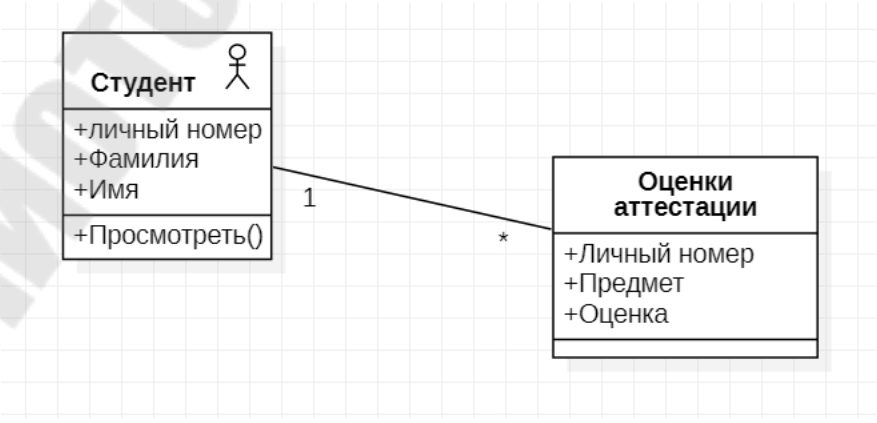


Рис. 3.2. Отношение ассоциации

Частным случаем ассоциации является отношение типа "часть/целое". Отношение такого типа называется агрегированием. В языке UML оно причислено к отношениям вида "имеет". Агрегирование изображается в виде ассоциации с незакрашенным ромбом со стороны целого. На рис. 3.3. приведен пример отношения агрегирования.

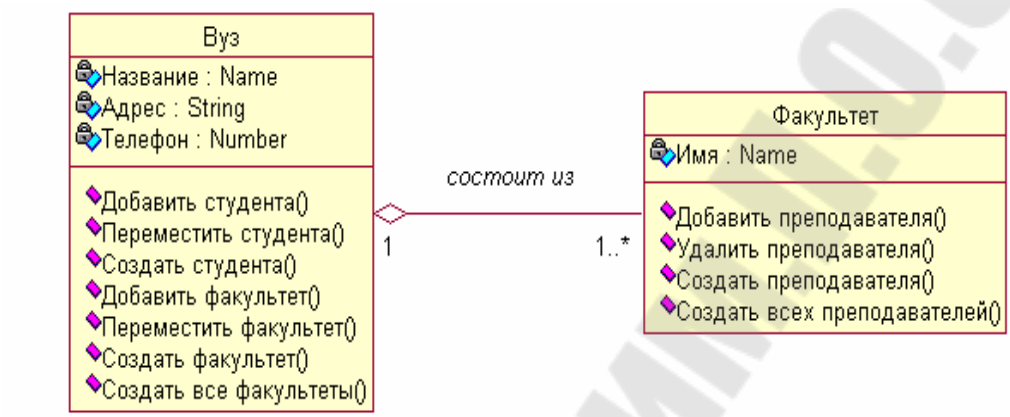


Рис. 3.3. Пример отношения агрегирования

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, т. е. с уничтожением целого уничтожаются и все его составные части (рис.3.4).

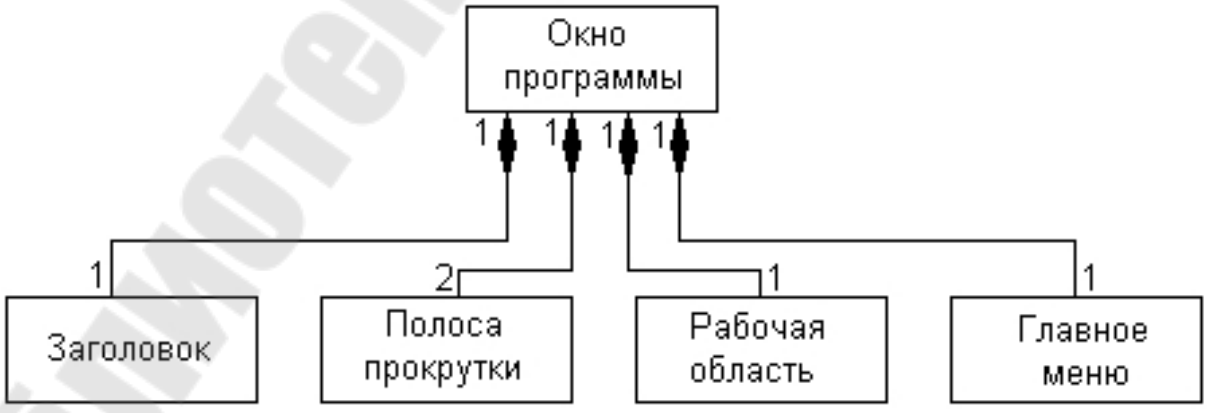


Рис. 3.4. Пример отношения композиции

Отношение обобщения является отношением между более общим элементом (родителем или предком) и более частным или специальным элементом (дочерним или потомком). Данное отношение может использоваться для представления взаимосвязей между пакетами, классами, вариантами использования и другими элементами языка UML (рис. 3.5).

Рис. 3.5. Отношение обобщения

На рис. 3.6 приведена диаграмма классов, в которой используются отношения ассоциации («Студент» - «Отчет по ЛР», «Преподаватель» - «Оценки аттестации») и отношения обобщения («Кафедра» - «Преподаватель»).

Стереотипы – это механизм, позволяющий разделять классы на категории. В языке UML определены три основных стереотипа классов:

- Boundary (граница),
- Entity (сущность),
- Control (управление).

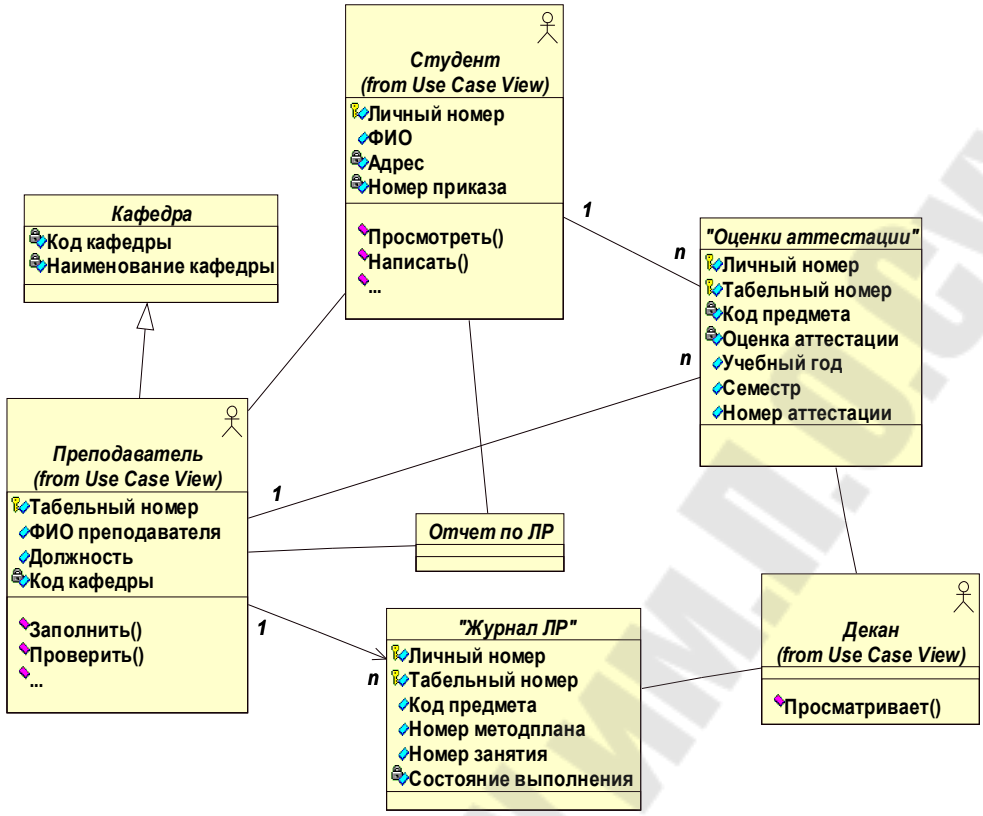
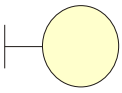


Рис. 3.6. Диаграмма классов с отношениями ассоциации и обобщения

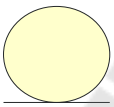
Ниже приведено описание стереотипов классов и их назначение.



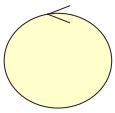
Граничный класс (boundary). Располагается на границе системы с внешней средой, но является составной частью системы.

Это экранные формы, отчеты, интерфейсы с аппаратурой (такой как принтеры или сканеры) и интерфейсы с другими системами.

Каждому взаимодействию между актером и прецедентом должен соответствовать, по крайней мере, один граничный класс.



Класс-сущность (entity)Содержит информацию, которая должна храниться постоянно и не уничтожаться с уничтожением объектов данного класса или прекращением работы моделируемой системы. Обычно для каждого класса-сущности создают таблицу в базе данных.



Управляющий класс (control)Отвечает за координацию действий других классов. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

На рис.3.7. приведен пример применения стереотипов в диаграмме классов.

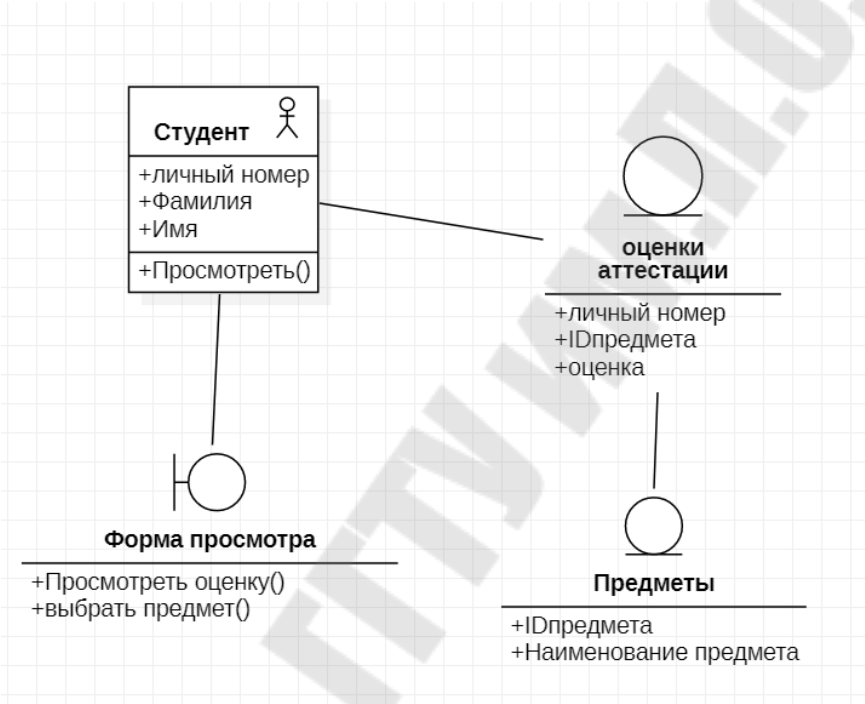


Рис. 3.7. Пример применения стереотипов в диаграмме классов

Практическая часть темы 3

Построение диаграммы классов осуществляется на основании диаграммы вариантов использования и технического задания, разработанного для выбранной предметной области. Задания для построения диаграммы классов приведено ниже.

1. Открыть существующий проект с разработанной диаграммой прецедентов, добавить в него новую диаграмму классов.
2. Определить объекты (сущности), привязав их к диаграмме прецедентов.
3. Дать имя классу для однотипной группы объектов, например объекты «Преподаватели» можно поместить в класс

«Преподаватель». Разработать перечень атрибутов для каждого класса, для каждого атрибута указать:

- квантора видимости атрибута,
- имени атрибута,
- типа значений атрибута,
- исходного значения атрибута (если есть необходимость).

4. Назначить атрибут – ключ (идентификатор объекта), например, для объекта «Преподаватель» – это может быть «Табельный номер преподавателя».
5. Указать операции над классом, например для класса «Преподаватель» – «Проверить()».
6. Построить отношения между классами на основе ассоциаций.
7. Определить направление и множественность, указав нижние и верхние границы.
8. Присвоить стереотипы классам, особое внимание уделить граничным классам, на основе которых будет строиться сценарий интерфейса пользователя.
9. Оформить отчет по лабораторной работе как приложение к техническому заданию.

Тема 4. Диаграммы последовательности и кооперации

Краткие теоретические сведения

Диаграммы взаимодействия являются моделями, описывающими поведение взаимодействующих групп объектов. Как правило, диаграмма взаимодействия охватывает поведение только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой в рамках данного варианта использования.

Данный вид диаграмм отражает следующие аспекты проектируемой системы:

- обмен сообщениями между объектами (в том числе в рамках обмена сообщениями со сторонними системами)
- ограничения, накладываемые на взаимодействие объектов
- события, инициирующие взаимодействия объектов.

Существует два вида диаграмм взаимодействия:

- диаграммы последовательности (sequence diagrams),

– диаграммы кооперации или сотрудничества (collaboration diagrams).

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

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

Одним из основных принципов ООП является способ информационного обмена между элементами системы, выражающийся в отправке и получении сообщений друг от друга. Таким образом, основные понятия диаграммы последовательности связаны с понятием **Объект и Сообщение.**

Рассмотрим основные компоненты диаграммы последовательности, одним из которых являются ***объекты***. На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные статические ассоциации с другими объектами. В качестве объектов могут выступать пользователи, инициирующие взаимодействие, классы, обладающие поведением в системе, программные компоненты, системы в целом (рис. 4.1.).

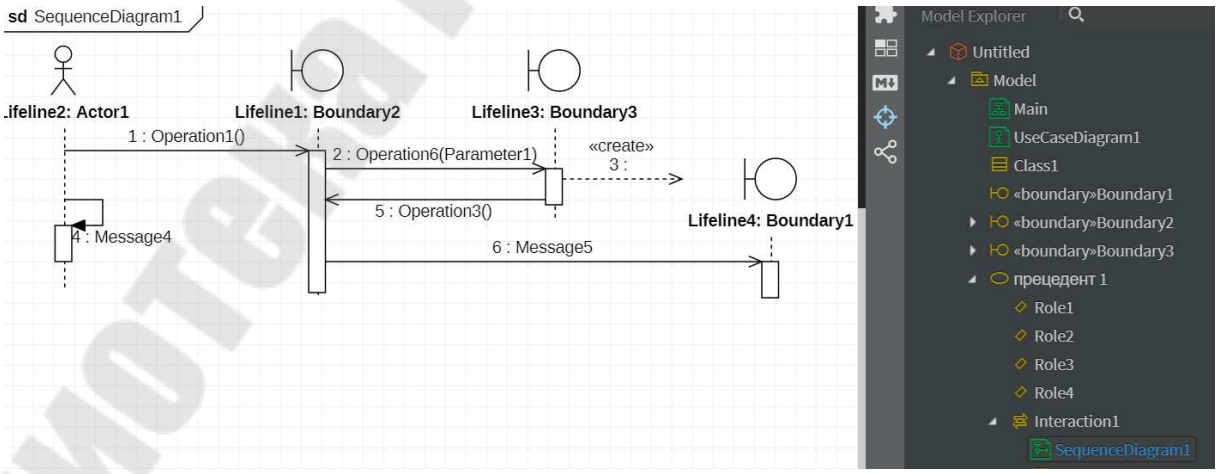


Рис. 4.1. Примеры объектов диаграммы последовательности

Имя объекта можно сформировать по следующему шаблону (таб.4.1.).

Таблица 4.1

Описание объектов

Наименование	Описание	Пример
Имя объекта : Имя класса	Известно имя объекта и имя класса	**Иванов: Студент**
: Имя класса	Анонимный объект	**: Студент**
Имя объекта	Имя класса известно	**Иванов**
Имя объекта :	Объект-сирота, имя класса неизвестно	**Иванов:**

Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. Время изменяется вдоль вертикального направления, начало отсчета находится вверху.

Диаграмма последовательности имеет два измерения: слева направо размещаются объекты, а сверху вниз отображается время. Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия. Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый очередностью или степенью активности объектов при взаимодействии друг с другом.

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

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

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

На диаграмме последовательности мы можем увидеть следующие аспекты сообщений:

- сообщения, побуждающие объект к действию;

- действия, которые вызываются сообщениями (методы) – зачастую это передача сообщения следующему объекты или возвращение определенных данных объекта;
- последовательность обмена сообщениями между объектами.

Каждое сообщение должно иметь имя по одному из следующих вариантов:

- произвольная строка текста (применяется на начальных стадиях проектирования или концептуальных диаграммах);
- указание стереотипа для некоторых стандартных действий;
- указание спецификации вызываемого метода объекта-получателя в формате:
  [переменная =] имя([список параметров]) [:возвращаемое значение].
  Виды сообщений приведены на рис. 4.2.

Рис. 4.2. Классификация видов сообщений диаграммы последовательности

Простое сообщение (simple) используется по умолчанию. Означает, что все сообщения выполняются в одном потоке управления. Это свойство задается добавляемому на диаграмму сообщению по умолчанию.

Синхронное (synchronous) - После передачи данного сообщения клиент ожидает ответа от объекта-приемника о результате выполнения соответствующей операции

Сообщение с отказом становиться в очередь (balking): После передачи данного сообщения объект-приемник отказывает клиенту в выполнении соответствующей операции, если он занят выполнением других операций

Сообщение с лимитированным временем ожидания (timeout): После передачи данного сообщения объект-приемник может поместить данное сообщение в очередь с ограниченным временем ожидания, если он занят выполнением других операций

Асинхронное сообщение (asynchronous): Клиент посылает данное сообщение и продолжает свою работу, не ожидая подтверждения от объекта-приемника о получении этого сообщения. При этом соответствующая операция может быть как выполнена, так и не выполнена

На диаграмме последовательности все сообщения упорядочены по времени своего возникновения в моделируемой системе. Сообщения, расположенные на диаграмме последовательности выше, инициируются раньше тех, которые расположены ниже.

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

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

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы «**X**».

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия, или состоянии пассивного ожидания сообщений от других объектов.

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

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

На рис. 4.3 приведен графический вид основных компонентов диаграммы последовательности.

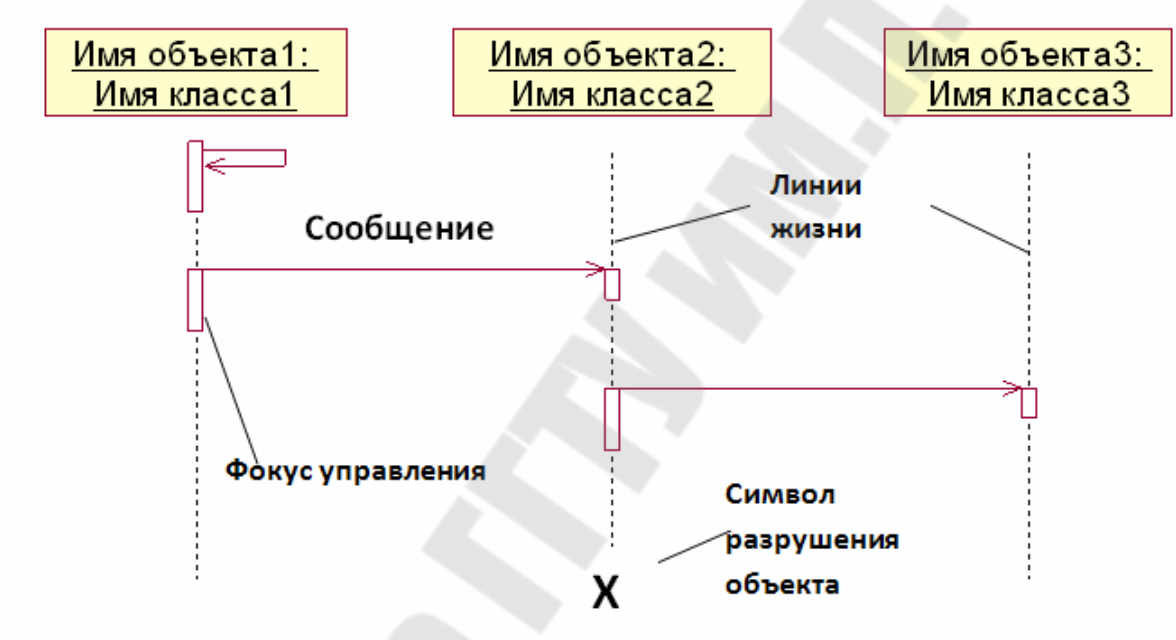


Рис. 4.3. Основные компоненты диаграммы последовательности

Если сообщение инициирует выполнение операции соответствующего класса, то эта операция должна быть выбрана из перечня операций класса после формирования стрелки сообщения (рис. 4.4).

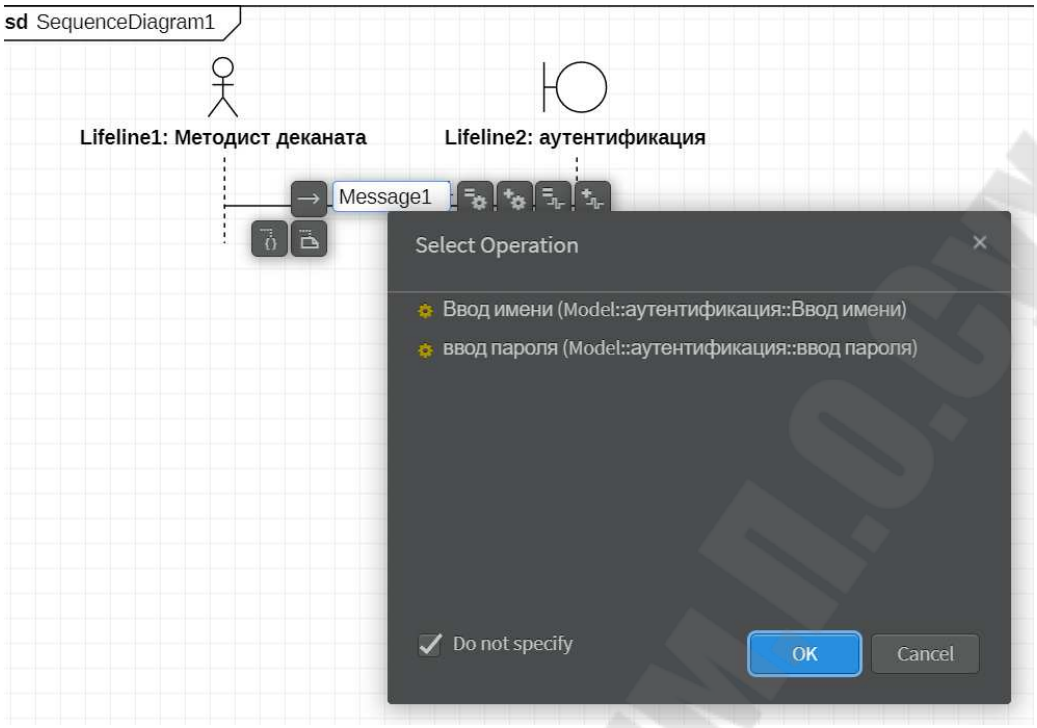


Рис. 4.4. Выбор операции для сообщения

Например, если в диаграмме классов бы создан граничный класс «Аутентификация», то в диаграмме последовательности для инициирования операции этого класса «Ввод имени» она должна быть выбрана в качестве операции сообщения (рис.4.5).

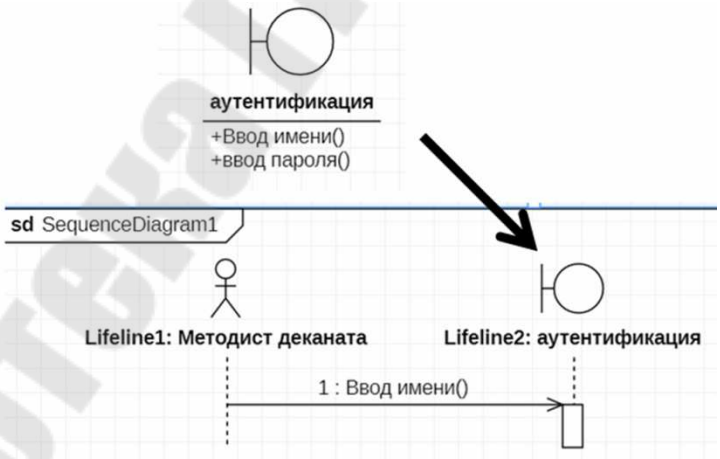


Рис. 4.5. Инициирование операции граничного класса

Пример сообщения с передачей параметров для объекта-получателя приведен на рис. 4.6.

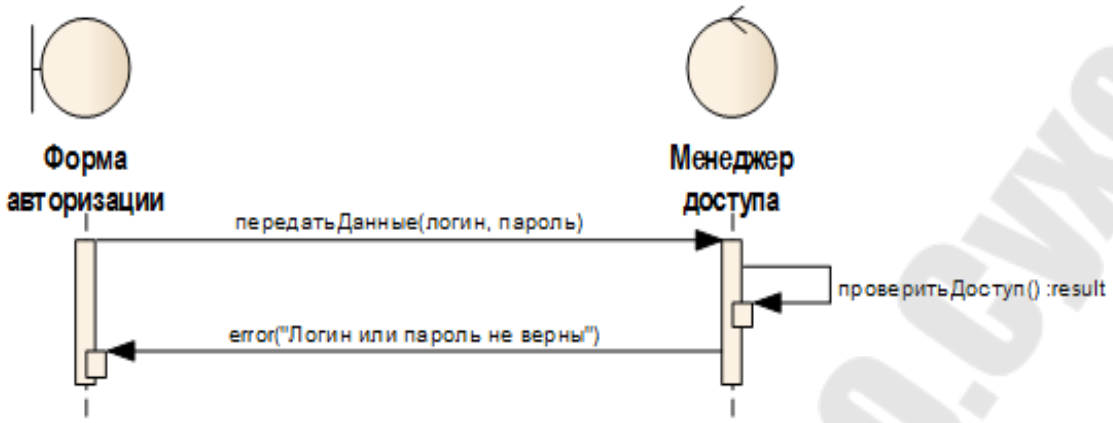


Рис. 4.6. Пример сообщения с передачей параметров для объекта-получателя

На рис. 4.7 приведен фрагмент общей диаграммы последовательности для прецедента «Формирование оценок аттестации студентов».

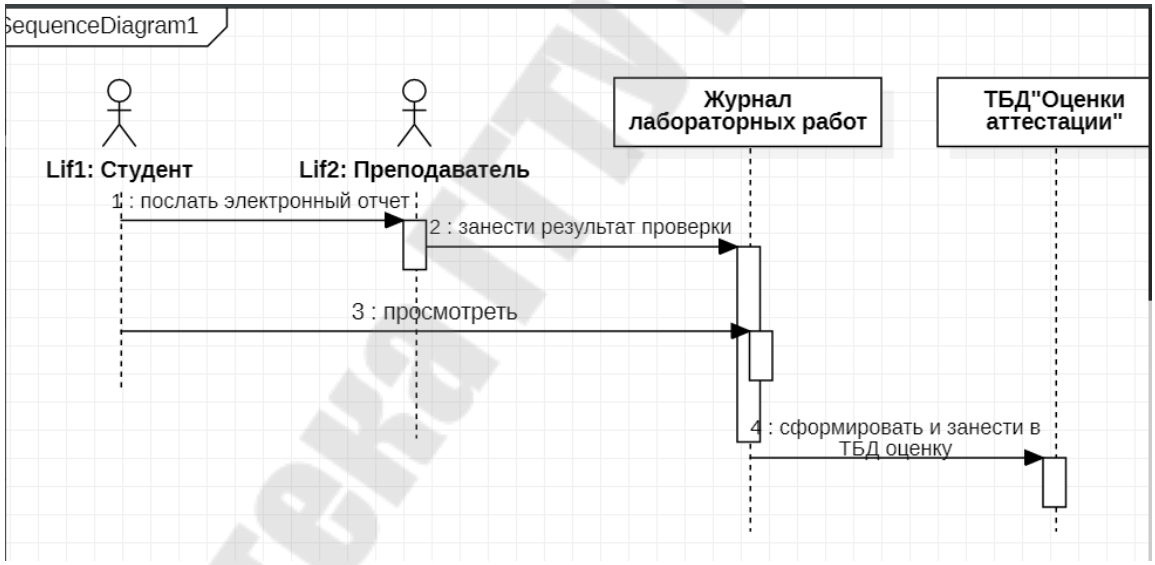


Рис. 4.7. Пример общей диаграммы последовательности

На рис. 4.8, 4.9 приведены примеры диаграмм последовательности с инициированием операций классов для отдельного прецедента.

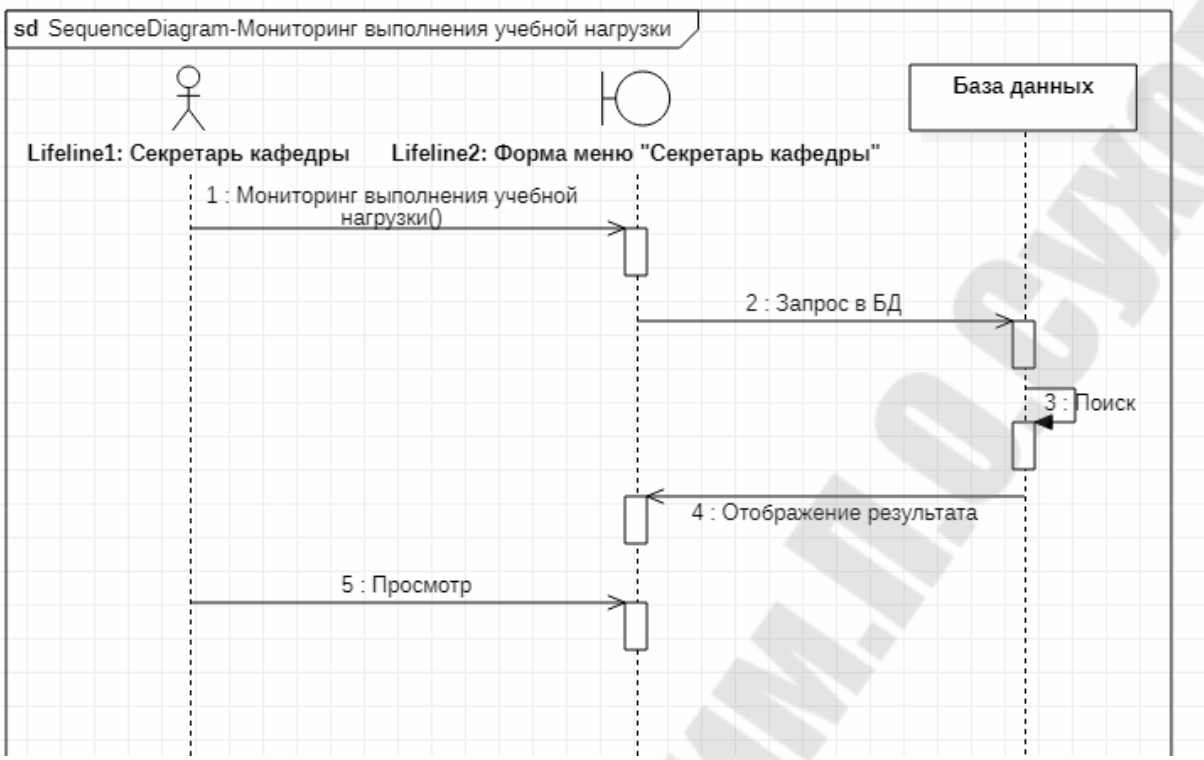


Рис. 4.8. Пример диаграммы последовательности с инициированием операций классов

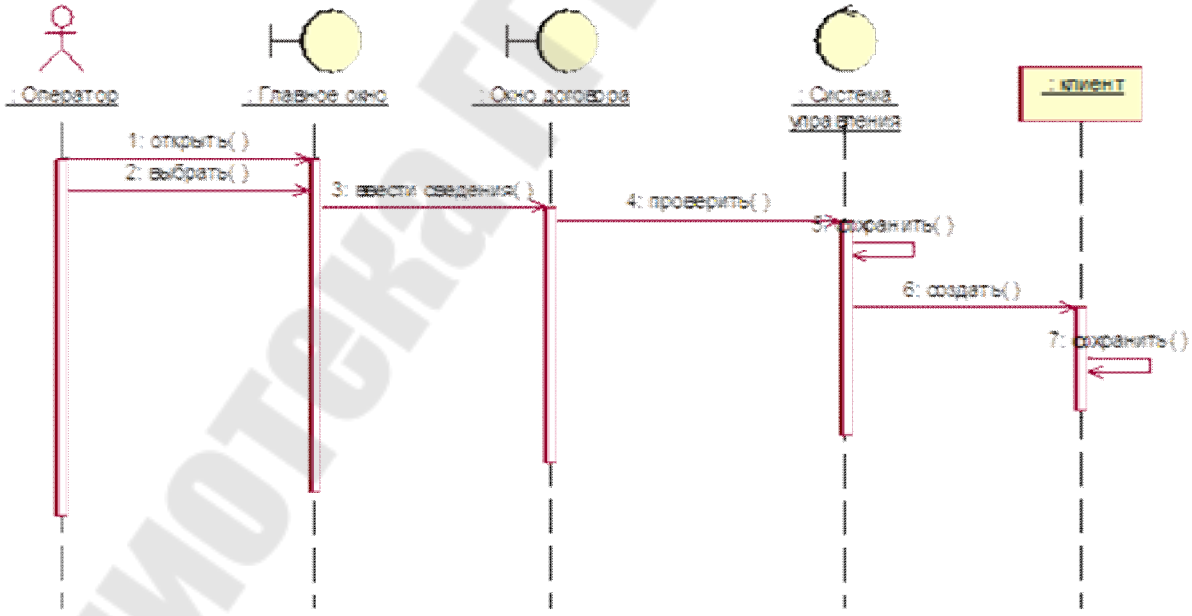


Рис. 4.9. Пример диаграммы последовательности с инициированием операций классов

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

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

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

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

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

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

На рис. 4.11 приведен пример диаграммы кооперации, разработанной на основе диаграммы последовательности, приведенной на рис. 4.10.

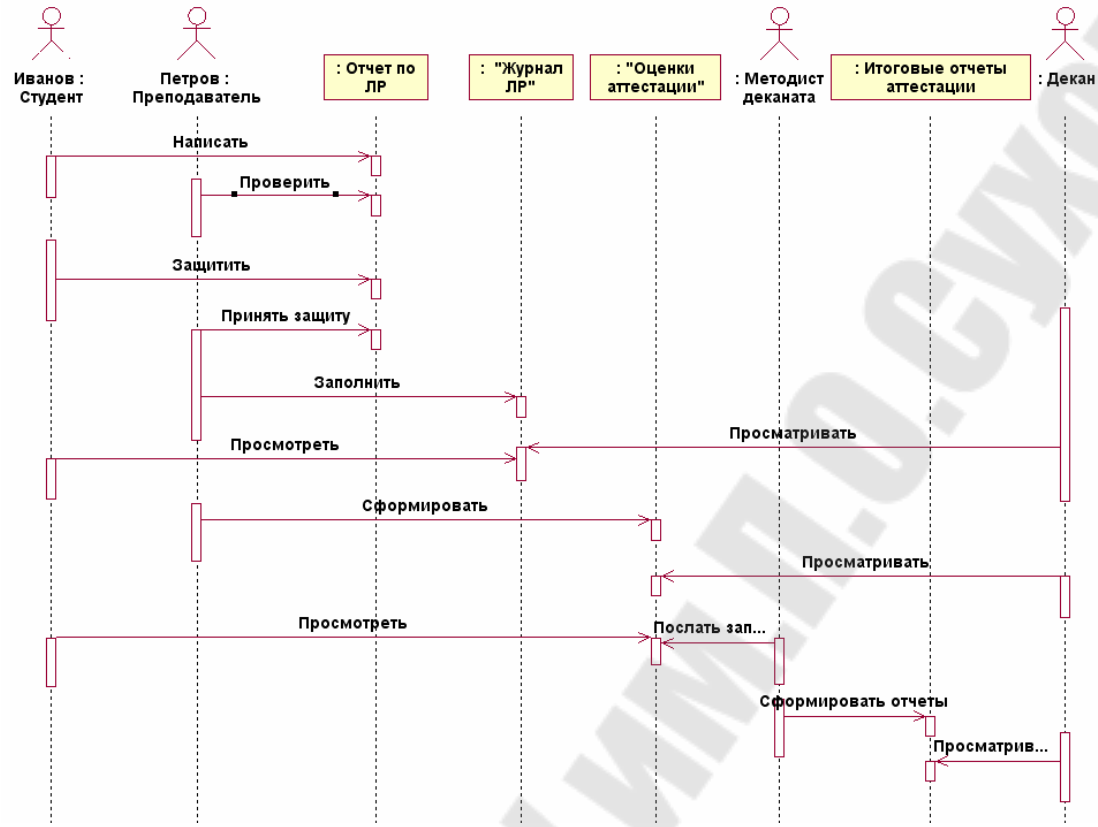


Рис. 4.10. Пример диаграммы последовательности

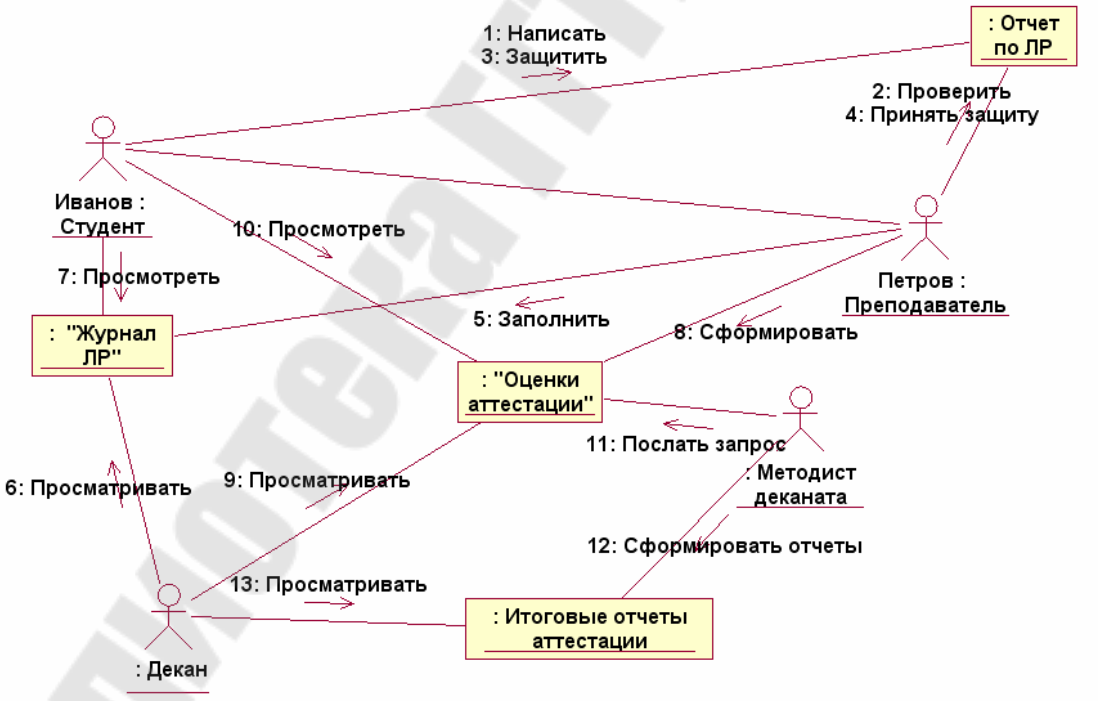


Рис. 4.11. Пример диаграммы кооперации

Практическая часть темы 4

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

Диаграмма кооперации строится полностью на основе уже разработанной диаграммы последовательности.

Порядок построения диаграмм последовательности и кооперации приведен ниже.

1. Открыть существующий проект с разработанной диаграммой прецедентов и диаграммой классов.
2. Определить объект, к которому привязана диаграмма последовательности (операция класса или вариант использования). Можно выбрать в качестве объекта проект целиком.
3. Создать диаграмму последовательности для объекта.
4. Создать линии жизни объектов.
5. Установить сообщения между объектами, используя там, где возможно, операции классов.
6. Присвоить имена новым сообщениям.
7. Пронумеровать сообщения.
8. Оформить отчет по лабораторной работе как приложение к техническому заданию.

Тема 5. Диаграммы состояний и деятельности

Краткие теоретические сведения
Диаграмма состояний

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

Каждая из них фиксирует внимание на отдельном аспекте функционирования системы.

В отличие от других диаграмм диаграмма состояний описывает процесс изменения состояний только одного класса, а точнее —

одного экземпляра определенного класса, т. е. моделирует все возможные изменения в состоянии конкретного объекта.

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

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

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

Состояние может быть задано в виде набора конкретных значений атрибутов класса или объекта, при этом изменение их отдельных значений будет отражать изменение состояния моделируемого класса или объекта (рис.5.1).

Рис. 5.1. Графическое обозначение состояния

Список внутренних действий в состоянии включает следующие значения:

- ***entry*** - действие, которое выполняется в момент входа в данное состояние (входное действие);
- ***exit*** - действие, которое выполняется в момент выхода из данного состояния (выходное действие);
- ***do*** - выполняющаяся деятельность ("do activity") в течение всего времени, пока объект находится в данном состоянии.

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

Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних

действий. В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени (рис.5.2).

Рис. 5.2. Графическое обозначение начального и конечного состояний

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

Пребывание моделируемого объекта в первом состоянии может сопровождаться выполнением некоторых действий, а переход во второе состояние будет возможен после завершения этих действий, а также после удовлетворения некоторых дополнительных условий. В этом случае говорят, что переход срабатывает или происходит срабатывание перехода.

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

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

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

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

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

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

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

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

В языке UML события играют роль стимулов, которые инициируют переходы из одних состояний в другие. В качестве событий можно рассматривать сигналы, вызовы, окончание фиксированных промежутков времени или моменты окончания выполнения определенных действий.

Имя события идентифицирует каждый отдельный переход на диаграмме состояний и может содержать строку текста, начинающуюся со строчной буквы. В этом случае принято считать переход **триггерным**, т. е. таким, который специфицирует событие-триггер.

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

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

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

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

На рис. 5.3. приведен пример диаграммы состояний и отображения ее объектов в дереве проекта.

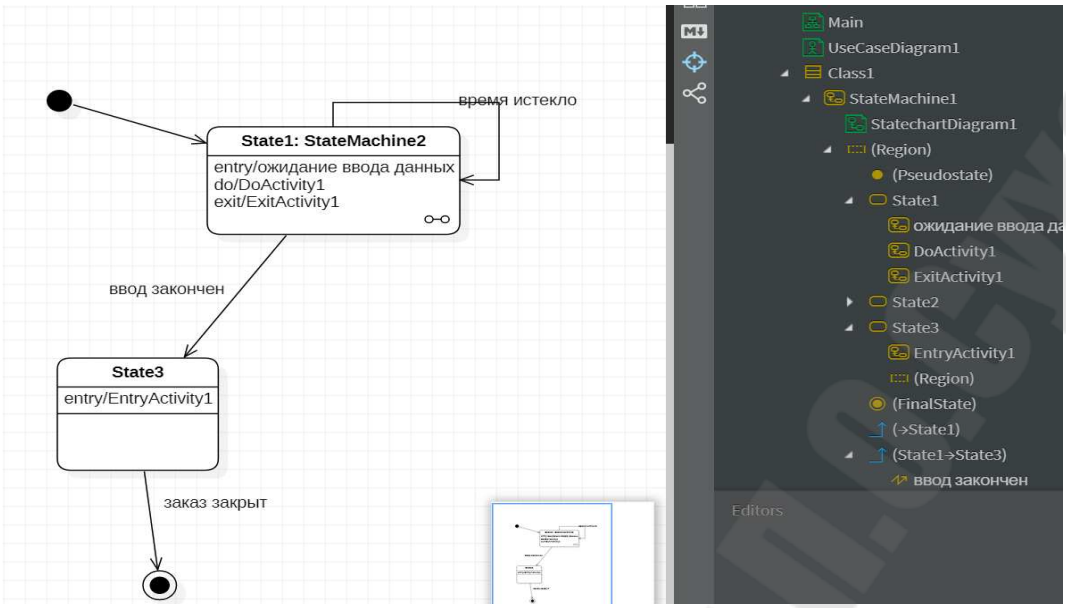


Рис. 5.3. Диаграмма состояний с внутренними действиями

Пример диаграммы состояний для объекта «Заказ» приведен на рис. 5.4.

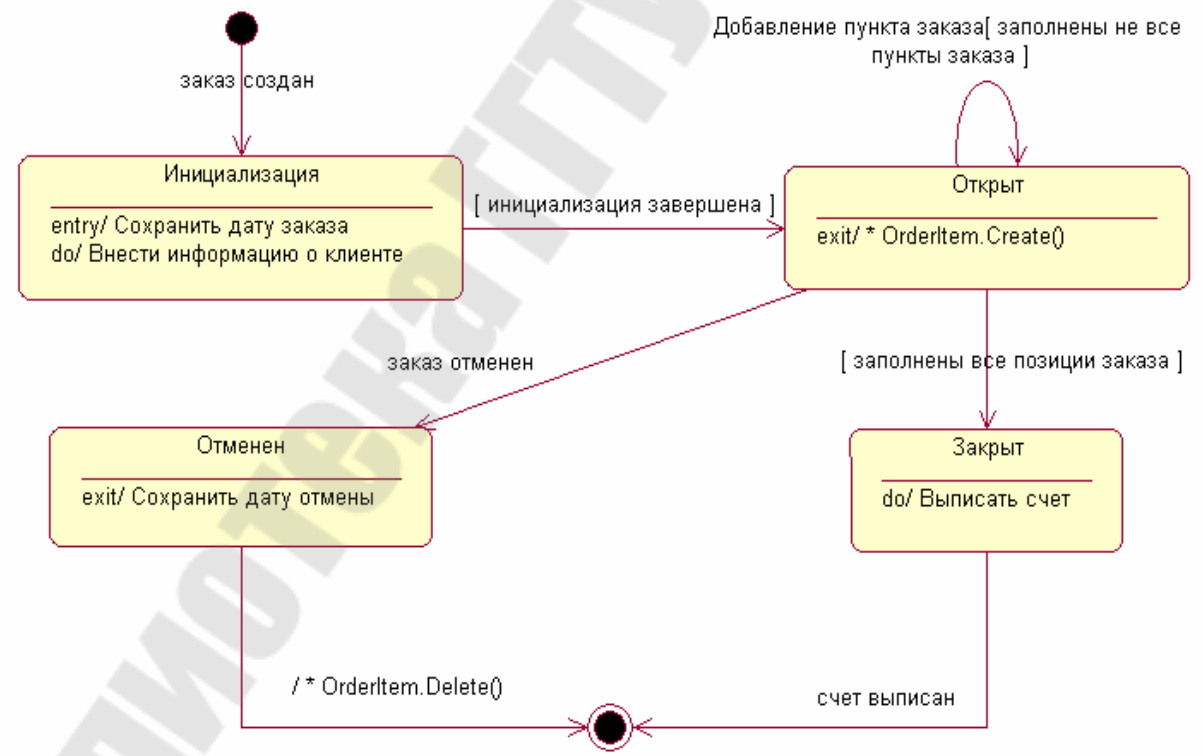


Рис. 5.4. Пример диаграммы состояний с триггерным переходом

Диаграмма 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.

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

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

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

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

Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы.

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

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

На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Состояние действия (action state) является специальным случаем состояния с некоторым входным действием и, по крайней мере, одним выходящим из состояния переходом. Этот переход неявно предполагает, что входное действие уже завершилось. Состояние действия не может иметь внутренних переходов, поскольку оно является элементарным.

Обычное использование состояния действия заключается в моделировании одного шага выполнения алгоритма (процедуры) или потока управления (рис.5.5).

Рис. 5.5. Примеры состояний действия: простое действие и выражение

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

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

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

Если последовательно выполняемая деятельность должна разделиться на альтернативные ветви в зависимости от значения

некоторого промежуточного результата, то в диаграмме применяется символ ветвления. Для отображения расширений сценария на диаграмме деятельности используются, так называемые узлы решения. **Узел решения** предназначен для определения правила ветвления и различных вариантов дальнейшего развития сценария. На рис. 5.6 приведен пример диаграммы деятельности с использованием узлов решения.

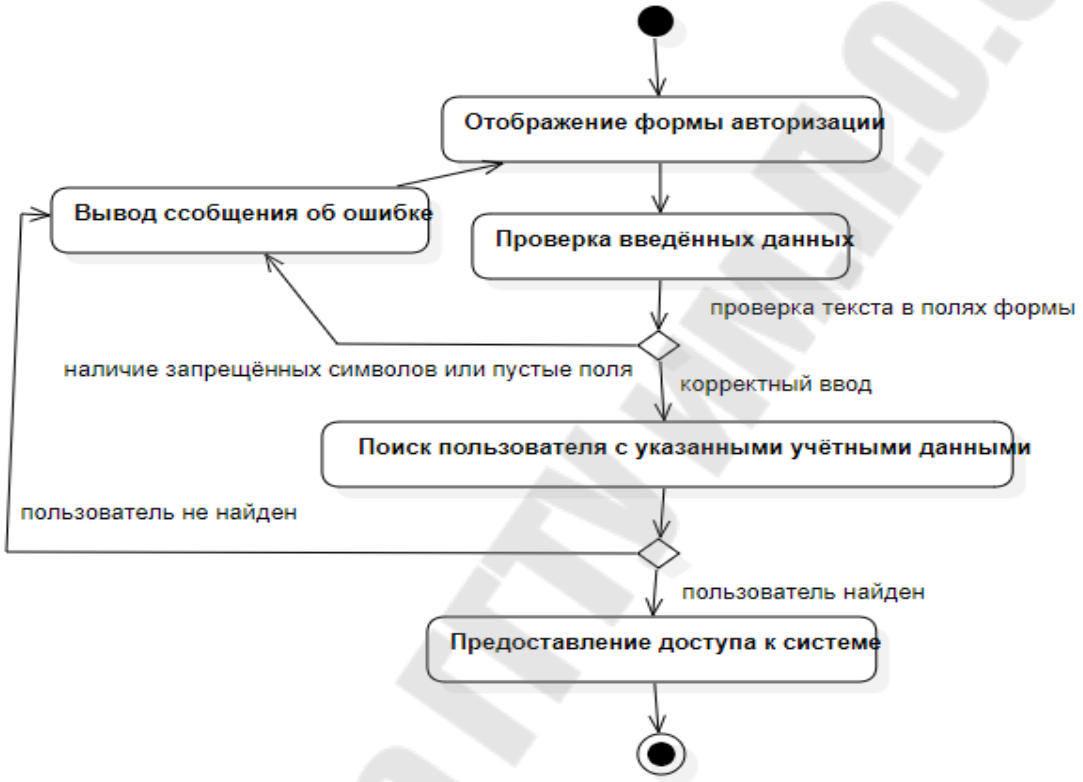


Рис. 5.6. Пример диаграммы деятельности с использованием узлов решения

Один из наиболее значимых недостатков обычных блок-схем или структурных схем алгоритмов связан с проблемой изображения параллельных ветвей отдельных вычислений. Поскольку распараллеливание вычислений существенно повышает общее быстродействие программных систем, необходимы графические примитивы для представления параллельных процессов. В языке UML для этой цели используется специальный символ для разделения и слияния параллельных вычислений или потоков управления. Таким символом является прямая черточка, аналогично обозначению перехода в формализме сетей Петри (рис. 5.7).

Рис. 5.7. Примеры разделения и слияния в диаграммах деятельности.

На рис. 5.8. приведен пример диаграммы деятельности с использованием слияния.

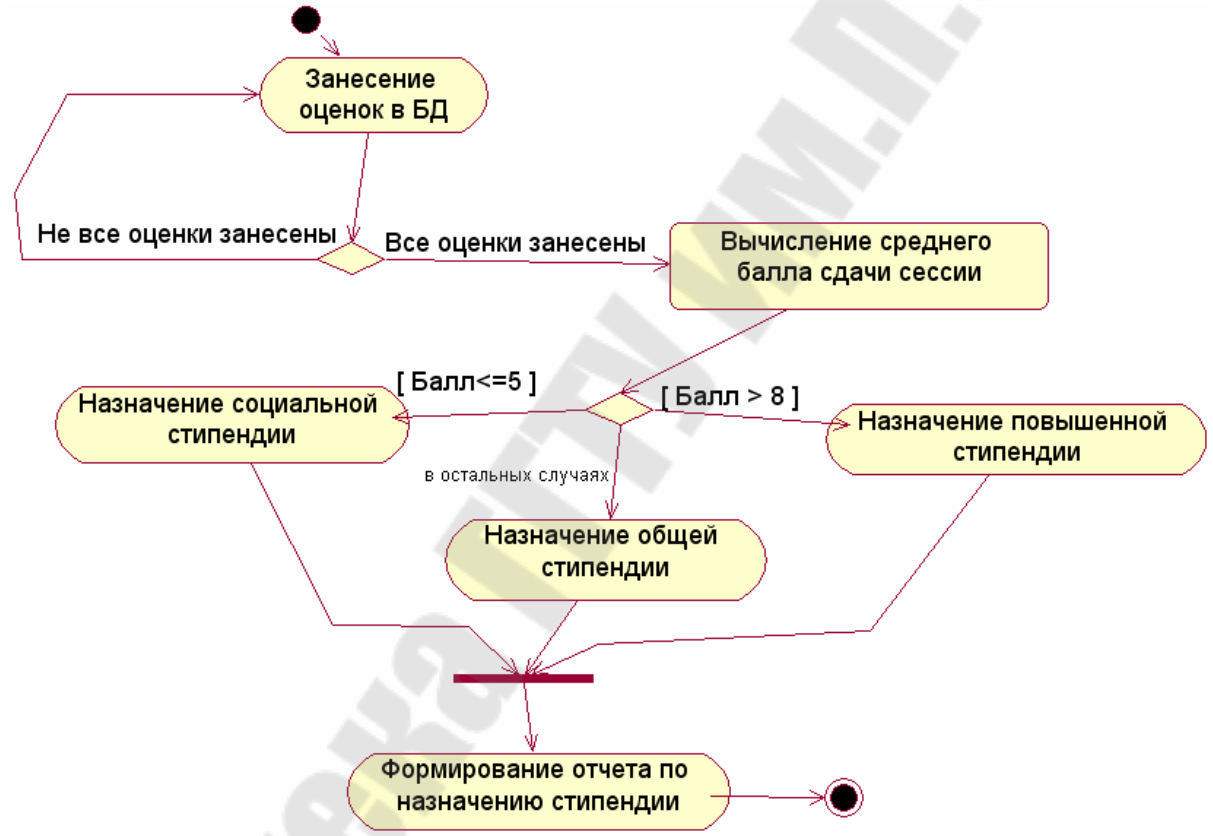


Рис. 5.8. Диаграмма деятельности с ветвлением и слиянием

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

Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название **дорожки – swimlanes** (рис.5.9).

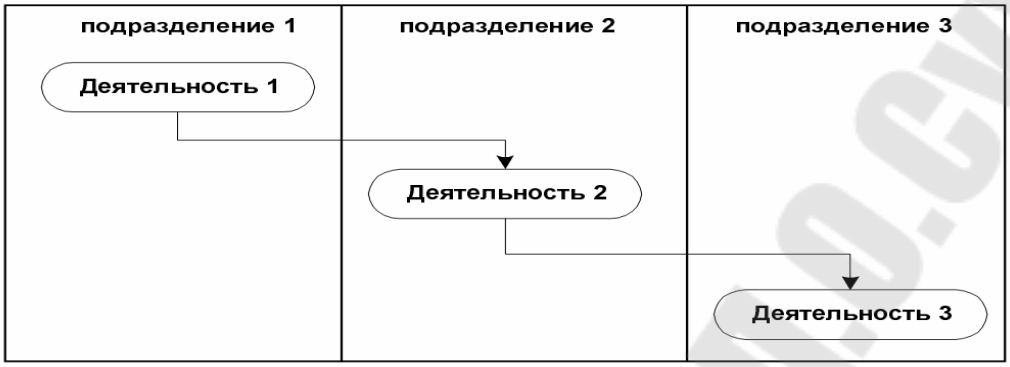


Рис. 5.9. Конструкция «Дорожки» в диаграмме деятельности

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

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

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

В качестве примера рассмотрим фрагмент диаграммы деятельности при расчете учебной нагрузки кафедры. Подразделениями, участвующими в процессе, являются методист учебного отдела, заведующий кафедрой, секретарь кафедры. 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 должно выполнять то или иное действие (рис. 5.10).

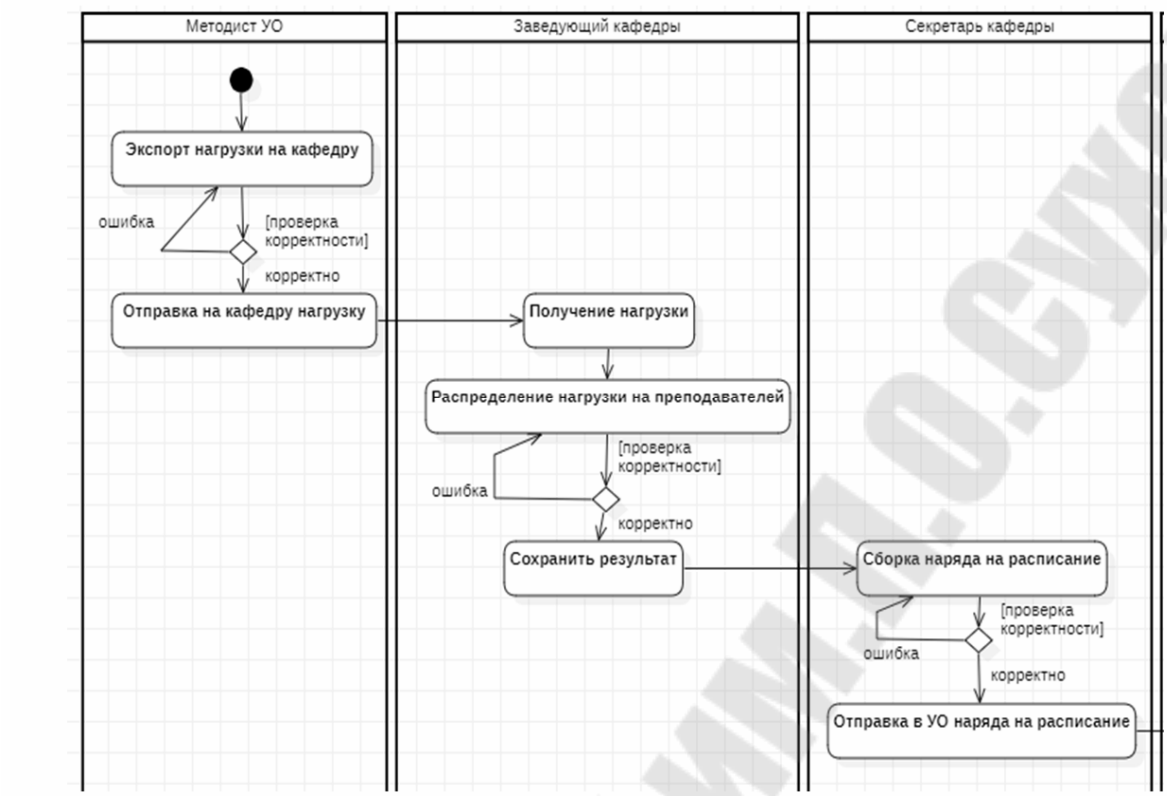


Рис. 5.10. Диаграмма деятельности с применением дорожек

Практическая часть темы 5

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

Последовательность разработки диаграммы состояний приведена ниже.

1. Открыть существующий проект с разработанными диаграммами прецедентов, классов и последовательности.
2. Дать имя диаграмме состояний.
3. Выбрать классы, для объектов которых будет строиться диаграмма состояний.
4. Разработать перечень состояний для объекта, учитывая начальное и конечное состояния.
5. Соединить состояния направленными переходами.

6. Выделить триггерные и нетриггерные переходы.
7. Разработать для триггерных переходов сторожевые условия и события.
8. Для каждого выбранного класса объектов разработать собственную диаграмму состояний.

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

1. Дать имя диаграмме деятельности.
2. Выбрать подразделения, для объектов которых будет строиться диаграмма деятельности.
3. Выделить зоны ответственности каждого подразделения и указать наименования подразделений на дорожках.
4. Для каждой дорожки разработать набор состояний действия.
5. Выделить ветвления и разработать переходы с использованием ветвлений.
6. Указать сторожевые условия на переходах, там, где это необходимо.
7. Выделить параллельные процессы и соединить нужные состояния действия с использованием графических символов раздела и слияния.
8. Оформить отчет по лабораторной работе как приложение к техническому заданию. Привести в отчете дерево всего проекта с раскрытыми вкладками.

Литература

1. Иванова В., Перерва А. Путь аналитика. Практическое руководство IT-специалиста. 2-е изд. — СПб.: Питер, 2015.
2. ГОСТ 34.602-89 Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы.
3. Иванов Д. Ю Унифицированный язык моделирования UML: Учеб. Пособие / Д. Ю. Иванов, Ф. А. Новиков – СПб.: Изд-во Политехн. ун-та, 2011.
4. Ларман К. Применение UML 2.0 и шаблонов проектирования. 3-е издание.: Пер. с англ. – М.: Издательский дом “Вильямс”, 2007.

5. Рамбо Д., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка – СПб.: Питер, 2007

Приложения

Приложение 1

Календарный план-график работ на сентябрь 2022

№ этапа	Наименование работы	Исполнитель	Срок сдачи	Выполнено
1	Подготовить описание предметной области и разработать организационную схему предприятия			
2				
…				
7	Скомпоновать из полученных материалов текст технического задания		20.09.22	

Приложение 2

Бланк титульного листа технического задания

СОГЛАСОВАНО

Директор ООО « »

_____

УТВЕРЖДАЮ
Заместитель генерального директора
по производству

_____ Трохова Т.А.

ТЕХНИЧЕСКОЕ ЗАДАНИЕ
НА ПРОГРАММНУЮ ПРОДУКЦИЮ

АВТОМАТИЗИРОВАННАЯ СИСТЕМА УПРАВЛЕНИЯ ПРЕДПРИЯТИЕМ (АСУП)

_____

ФУНКЦИОНАЛЬНОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
АСУП _____

(ШИФР: _____)

от Исполнителя

_____

_____

от Заказчика

_____

_____

г. Гомель, 2022

Приложение 3

Описание задач для подсистемы «Расписание»

«Расписание – кафедра»

Основные функции:

- формирование таблицы БД «Наряд на расписание»;
- формирование расписания зачетов;
- формирование расписания приема курсовых работ;
- автоматизированное формирование расписания преподавателей на семестр.

Входная информация:

- таблица БД «Нагрузка преподавателя»;
- таблица БД «Расписание»

Выходная информация

- таблица БД «Наряд на расписание»
- расписание зачетов
- расписание приема курсовых работ
- индивидуальные листки расписания преподавателей на семестр
- общее расписание по кафедре

Нормативно-справочная информация

- справочник аудиторий;
- справочник факультетов;
- справочник групп;
- справочник студентов;
- справочник кафедр;
- справочник преподавателей;
- справочник предметов.

«Расписание – учебный отдел»

Основные функции:

- формирование таблицы БД «Расписание»;
- формирование результирующего расписания учебных групп;
- автоматизированное формирование справки о загрузке аудиторий;
- автоматизированное формирование отчета о применении проектора.

Входная информация:

- таблица БД «Наряд на расписание»

Выходная информация

- таблица БД «Расписание»;
- расписание по группам факультетов;
- справка о загрузке аудиторий;
- справка о применении проектора.

Нормативно-справочная информация

- справочник аудиторий;
- справочник факультетов;
- справочник групп;
- справочник студентов;
- справочник кафедр;
- справочник преподавателей;
- справочник предметов.

«Учебная нагрузка – кафедра»

Основные функции:

- формирование таблицы БД «Учебная нагрузка преподавателей»;
- формирование заявки на количество ставок от преподавателей;
- автоматизированное формирование отчета об учебной нагрузке по преподавателям;
- автоматизированное формирование отчета об учебной нагрузке кафедры.

Входная информация:

- таблица БД «Учебный план по специальностям»
- таблица БД «Учебная нагрузка общая»
- штатное расписание на кафедру;
- средняя нагрузка на кафедру;

Выходная информация

- таблица БД «Учебная нагрузка по преподавателям»;
- отчет об учебной нагрузке по преподавателям;
- отчета об учебной нагрузке по кафедре;

Нормативно-справочная информация

- справочник видов нагрузки;
- справочник факультетов;
- справочник групп;
- справочник кафедр;
- справочник преподавателей;
- справочник предметов.

«Учебная нагрузка - преподаватель»

Основные функции:

- формирование таблицы БД «Индивидуальный план преподавателя»;
- формирование справки о выполнении нагрузки преподавателя;
- автоматизированное заполнение индивидуального плана.

Входная информация:

- таблица БД «Учебная нагрузка преподавателя»
- таблица БД «План-график учебного процесса»
- индивидуальный план (бумажный).

Выходная информация

- таблица БД «Индивидуальный план преподавателя»;
- отчет о выполнении учебной нагрузки преподавателем.

Нормативно-справочная информация

- справочник видов нагрузки;
- справочник факультетов;
- справочник групп;
- справочник кафедр;
- справочник преподавателей;
- справочник предметов.

Приложение 4

Варианты индивидуальных заданий

1**. Спортивный комплекс**

- игровые залы, стадион;
- соревнования, тренировки;
- гостиница.

2. **Деканат**

- кафедры;
- работа декана, зам. декана;
- секретари.

3**. Ветеринарная клиника**

- поликлиника, прием животных;
- аптека;
- стационар.

4. **Птицефабрика**

- производство,
- склад готовой продукции,
- магазины.

5. **Охранная фирма**

- наладка охранного оборудования,
- отдел охраны,
- отдел вооружения.

6. **Кафедра**

- работа зав.кафедрой,
- работа секретаря,
- работа преподавателей,

7. **Страховая компания**

- работа менеджеров,
- отдел оценки,
- работа страховых агентов.

8. **Фабрика игрушек**

- производство,
- склад материалов и готовой продукции,
- магазины.

9. **Учебный отдел**

- диспетчеры расписания,
- специалисты учебно-методического отдела (учебная нагрузка),

- отдел практики.

10. **Фирма по трудоустройству**

- отдел работы с клиентами,
- департаменты подбора персонала по направлениям,
- дополнительные услуги.

11. **Склад нефтепродуктов**

- работа с поставщиками,
- работа с АЗС,
- работа с автотранспортным предприятием.

СОДЕРЖАНИЕ

Трохова Татьяна Анатольевна

ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Практикум по выполнению лабораторных работ для студентов специальности 1-40 04 01 «Информатика и технологии программирования» дневной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 24.01.24.
Рег. № 65Е.
http://www.gstu.by