Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Магнитогорский государственный технический университет
им. Г.И. Носова»
Многопрофильный колледж

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

для студентов специальностей
09.02.07 Информационные системы и программирование

ПМ.05. Проектирование и разработка информационных систем

МДК.05.02 Разработка кода информационных систем

Магнитогорск, 2020

ОДОБРЕНО
Предметно-цикловой комиссией
«Информатики и вычислительной техники»
Председатель И.Г.Зорина
Протокол № 7 от 17.02.2020г.

Методической комиссией МпК

Протокол №3 от «26» февраля 2020г

Составитель:
преподаватель ФГБОУ ВО «МГТУ им. Г.И. Носова» МпК Юлия Юрьевна Дорохина

Методические указания по выполнению практических работ разработаны на основе рабочей программы учебной дисциплины ПМ.05. Проектирование и разработка информационных систем.

Содержание практических работ ориентировано на подготовку обучающихся к освоению профессионального модуля программы подготовки специалистов среднего звена по специальности 09.02.07 Информационные системы и программирование и овладению общими компетенциями.

Методические указания по выполнению практических работ разработаны на основе рабочей программы ПМ.05. Проектирование и разработка информационных систем, МДК.05.02 Разработка кода информационных систем

Содержание практических работ ориентировано на формирование общих и профессиональных компетенций по программе подготовки специалистов среднего звена по специальности 09.02.07 Информационные системы и программирование.

СОДЕРЖАНИЕ

1 ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Состав и содержание практических занятий направлены на реализацию Федерального государственного образовательного стандарта среднего профессионального образования.

Ведущей дидактической целью практических занятий является формирование профессиональных практических умений (умений выполнять определенные действия, операции, необходимые в последующем в профессиональной деятельности) или учебных практических умений (умений решать задачи по математике, физике, химии, информатике и др.), необходимых в последующей учебной деятельности.

В соответствии с рабочей программой учебной дисциплины ПМ 05. «Проектирование и разработка информационных систем», МДК.05.02 Разработка кода информационных систем предусмотрено проведение практических занятий. В рамках практического занятия обучающиеся могут выполнять одну или несколько лабораторных или практических работ.

В результате их выполнения, обучающийся должен:

уметь:

- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств;
- использовать алгоритмы обработки информации для различных приложений;
- решать прикладные вопросы программирования и языка сценариев для создания программ;
- создавать и управлять проектом по разработке приложения;
- проектировать и разрабатывать систему по заданным требованиям и спецификациям;
- работать с инструментальными средствами обработки информации;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;
- распознавать задачу и/или проблему в профессиональном и/или социальном контексте;
- анализировать задачу и/или проблему и выделять её составные части;
- определять задачи для поиска информации;
- определять необходимые источники информации;
- определять и выстраивать траектории профессионального развития и самообразования;
- применять исследовательские приемы и навыки, чтобы быть в курсе последних отраслевых решений;
- взаимодействовать с коллегами, руководством, клиентами в ходе профессиональной деятельности;
- понимать требования и оправдывать ожидания клиентов/работодателя;
- применять техники и приемы эффективного общения в профессиональной деятельности;
- использовать навыки устного общения в профессиональной деятельности;
- применять средства информационных технологий для решения профессиональных задач;
- использовать современное программное обеспечение;
- понимать общий смысл четко произнесенных высказываний на известные темы (профессиональные и бытовые);
- участвовать в диалогах на знакомые общие и профессиональные темы;
- читать, понимать и находить необходимые технические данные и инструкции в руководствах в любом доступном формате;

- применять знания по финансовой грамотности для профессиональной деятельности и в повседневной жизни;

- выявлять достоинства и недостатки коммерческой идеи;

Содержание лабораторных и практических работ ориентировано на подготовку обучающихся к освоению профессионального модуля программы подготовки специалистов среднего звена по специальности и овладению ***профессиональными компетенциями:***

- ПК 5.1. Собирать исходные данные для разработки проектной документации на информационную систему.
- ПК 5.2. Разрабатывать проектную документацию на разработку информационной системы в соответствии с требованиями заказчика.
- ПК 5.3. Разрабатывать подсистемы безопасности информационной системы в соответствии с техническим заданием.
- ПК 5.4. Производить разработку модулей информационной системы в соответствии с техническим заданием.

А также формированию ***общих компетенций***:

- ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.
- ОК 02. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.
- ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие.
- ОК 04. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.
- ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.
- ОК 06. Проявлять гражданско-патриотическую позицию, демонстрировать осознанное поведение на основе традиционных общечеловеческих ценностей.
- ОК 07. Содействовать сохранению окружающей среды, ресурсосбережению, эффективно действовать в чрезвычайных ситуациях.
- ОК 08. Использовать средства физической культуры для сохранения и укрепления здоровья в процессе профессиональной деятельности и поддержания необходимого уровня физической подготовленности.
- ОК 09. Использовать информационные технологии в профессиональной деятельности.
- ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языках.
- ОК 11. Планировать предпринимательскую деятельность в профессиональной сфере

Выполнение обучающихся лабораторных и практических работ по учебной дисциплине ПМ 05. «Проектирование и разработка информационных систем», МДК.05.02 Разработка кода информационных систем направлено на:

- обобщение, систематизацию, углубление, закрепление, развитие и детализацию полученных теоретических знаний по конкретным темам учебной дисциплины;

- формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;

- развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных, конструктивных и др.;

- выработку при решении поставленных задач профессионально значимых качеств, таких как самостоятельность, ответственность, точность, творческая инициатива.

Практические занятия проводятся после соответствующей темы, которая обеспечивает наличие знаний, необходимых для ее выполнения.

2 ПЕРЕЧЕНЬ ПРАКТИЧЕСКИХ ЗАНЯТИЙ

МДК.05.02 РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ

Разделы/темы	Темы практических/лабораторных занятий	Количество часов	Требования ФГОС СПО (уметь)
Раздел 2. Инструментарий и технологии разработки кода информационных систем		**83**	У 5.1, У 5.2, У5.3, У5.4
Тема 5.2.1. Основные инструменты для создания, исполнения и управления информационной системой	Лабораторная работа №1 Построение диаграммы Вариантов использования и диаграммы. Последовательности и генерация кода	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа №2 Построение диаграммы Кооперации и диаграммы Развертывания и генерация кода	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа №3 Построение диаграммы Деятельности, диаграммы Состояний и диаграммы Классов и генерация кода	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа №4 Построение диаграммы компонентов и генерация кода	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа №5 Построение диаграмм потоков данных и генерация кода	3	У 5.1, У 5.2, У5.3, У5.4
Тема 5.2.2. Разработка и модификация информационных систем	Практическая работа №1 Обоснование выбора технических средств	4	У 5.1, У 5.2, У5.3, У5.4
	Практическая работа №2 Стоимостная оценка проекта	4	У 5.1, У 5.2, У5.3, У5.4
	Практическая работа №3 Построение и обоснование модели проекта	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 6 Установка и настройка системы контроля версий с разграничением ролей	4	У 5.1, У 5.2, У5.3, У5.4
	Практическая работа №4 Проектирование и разработка интерфейса пользователя	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 7 Разработка графического интерфейса пользователя	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 8 Реализация алгоритмов обработки числовых данных. Отладка приложения	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 9 Реализация алгоритмов поиска. Отладка приложения	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 10 Реализация обработки табличных данных. Отладка приложения	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 11 Разработка	4	У 5.1, У 5.2,

	и отладка генератора случайных символов		У5.3, У5.4
	Лабораторная работа № 12 Разработка приложений для моделирования процессов и явлений. Отладка приложения	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 13 Интеграция модуля в информационную систему	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 14 Программирование обмена сообщениями между модулями	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 15 Организация файлового ввода-вывода данных	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 16 Разработка модулей экспертной системы	4	У 5.1, У 5.2, У5.3, У5.4
	Лабораторная работа № 17 Создание сетевого сервера и сетевого клиента.	4	У 5.1, У 5.2, У5.3, У5.4
ИТОГО		**83**	

3 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

МДК.05.02 02 Разработка кода информационных систем

Тема 5.2.1. Основные инструменты для создания, исполнения и управления информационной системой

Лабораторная работа № 1 Построение диаграммы Вариантов использования и диаграммы Последовательности и генерация кода

Цель: ознакомиться с методологией моделирования диаграммы Вариантов и диаграммы Последовательности на основе языка UML

Выполнив работу, Вы будете:

уметь:

- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств.
- проектировать и разрабатывать систему по заданным требованиям и спецификациям.
- использовать методологию языка UML для моделирования диаграмм Вариантов и Последовательности

Материальное обеспечение:

Персональный компьютер.

Задание:

1. Ознакомиться с теоретическим материалом.
2. Ознакомиться с методологией построения диаграмм вариантов и последовательности на основе языка UML.
3. Постройте диаграмму вариантов использования для выбранной информационной системы
4. Выполните построение диаграммы последовательности для выбранной информационной системы
5. Сдайте выполненную работу.

Краткие теоретические сведения:

Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее общей концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Работа над проектом начинается с общего анализа проблемы и построения диаграммы ***вариантов использования*** (use case diagram), которая отражает функциональное назначение проектируемой системы.

Диаграммы вариантов использования разрабатывают с целью:

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

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или

действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик.

В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

Конструкция или стандартный элемент языка UML вариант использования применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст называется примечанием или сценарием.

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

Рис. 1. Графическое обозначение варианта использования

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

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

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

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

Каждый выполняемый вариантом использования метод реализуется как неделимая транзакция, т. е. выполнение сервиса не может быть прервано никаким другим экземпляром варианта 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. При этом актеры служат для обозначения согласованного множества ролей, которые могут играть пользователи в процессе взаимодействия с проектируемой системой. Каждый актер может рассматриваться как некая отдельная роль относительно конкретного варианта использования. Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается конкретное имя актера (рис. 2).

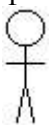

Рис. 2. Графическое обозначение актера

В некоторых случаях актер может обозначаться в виде прямоугольника класса с ключевым словом "актер" и обычными составляющими элементами класса. Имена актеров должны записываться заглавными буквами и следовать рекомендациям использования имен для типов и классов модели. При этом символ отдельного актера связывает соответствующее описание актера с конкретным именем. Имена абстрактных актеров, как и других абстрактных элементов языка UML, рекомендуется обозначать курсивом. Имя актера должно быть достаточно информативным. Для этой цели подходят наименования должностей в компании (например, продавец, кассир, менеджер, президент). Не рекомендуется давать актерам имена собственные или названия моделей конкретных устройств, так как одно и то же лицо может выступать в нескольких ролях и, соответственно, обращаться к различным сервисам системы. Примерами актеров могут быть: клиент банка, банковский служащий, продавец магазина, менеджер отдела продаж, пассажир авиарейса, водитель автомобиля, администратор гостиницы, сотовый телефон и другие сущности, имеющие отношение к концептуальной модели соответствующей предметной области.

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

Так как в общем случае актер всегда находится вне системы, его внутренняя структура никак не определяется. Для актера имеет значение только его внешнее представление, т. е. то, как он воспринимается со стороны системы. Актеры взаимодействуют с системой посредством передачи и приема сообщений от 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 или классами. Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами.

Примечания (notes) в языке UML предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. В качестве такой информации могут быть комментарии разработчика (например, дата и версия разработки диаграммы или ее отдельных компонентов), ограничения (например, на значения отдельных связей или экземпляры

сущностей) и помеченные значения. Применительно к диаграммам вариантов использования примечание может носить самую общую информацию, относящуюся к общему контексту системы.

Графически примечания обозначаются прямоугольником с "загнутым" верхним правым уголком (рис. 3). Внутри прямоугольника содержится текст примечания. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия. Если примечание относится к нескольким элементам, то от него проводятся, несколько линий.

Рис. 3.Примеры примечаний в языке UML

Отношения на диаграмме вариантов использования. Между компонентами 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. Следует заметить, что два варианта использования, определенные для одной и той же сущности, не могут взаимодействовать друг с другом, поскольку каждый из них самостоятельно описывает законченный вариант использования этой сущности. Более того, варианты использования всегда предусматривают некоторые сигналы или сообщения, когда взаимодействуют с актерами за пределами системы. В то же время могут быть определены другие способы для взаимодействия с элементами внутри системы.

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

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

При этом общие свойства вариантов использования могут быть представлены тремя различными способами, а именно с помощью отношений расширения, обобщения и включения.

Отношение ассоциации является одним 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 оно служит для обозначения специфической роли актера в отдельном варианте использования. Это отношение устанавливает, какую конкретную роль играет актер при взаимодействии с экземпляром варианта использования. На диаграмме вариантов использования, так же, как и на других диаграммах, отношение ассоциации обозначается сплошной линией между актером и вариантом использования. Эта линия может иметь дополнительные условные обозначения, такие, например, как имя и кратность (рис. 4).

Рис. 4.Пример графического представления отношения ассоциации между актером и вариантом использования

Отношение расширения определяет взаимосвязь экземпляров отдельного варианта использования с более общим вариантом, свойства которого определяются на основе способа совместного объединения данных экземпляров. В метамодели 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"), как показано на рис. 5.

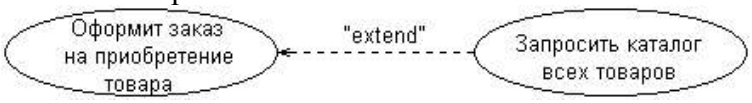

Рис. 5 Пример графического изображения отношения расширения между вариантами использования

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

В представленном выше примере (рис. 5) при оформлении заказа на приобретение товара только в некоторых случаях может потребоваться предоставление клиенту каталога всех товаров. При этом условием расширения является запрос от клиента на получение каталога товаров. Очевидно, что после получения каталога клиенту необходимо некоторое время на его изучение, в течение которого оформление заказа приостанавливается. После ознакомления с каталогом клиент решает либо в пользу выбора отдельного товара, либо отказа от покупки вообще. Сервис или вариант использования "Оформить заказ на приобретение товара" может отреагировать на выбор клиента уже после того, как клиент получит для ознакомления каталог товаров.

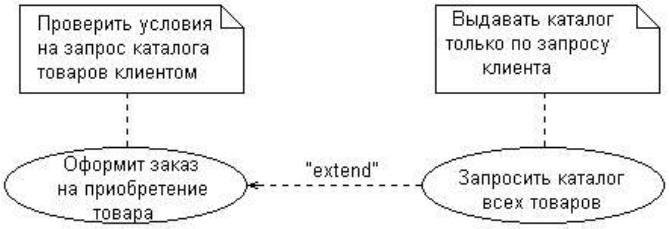

Рис. 6 Графическое изображение отношения расширения с примечаниями условий выполнения вариантов использования

Отношение обобщения служит для указания того факта, что некоторый вариант использования А может быть обобщен до варианта использования В. В этом случае вариант А будет являться специализацией варианта В. При этом В называется предком или родителем по отношению А, а вариант А - потомком по отношению к варианту использования В. Следует подчеркнуть, что 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 (рис. 7). Эта линия со стрелкой имеет специальное название - стрелка "обобщение".

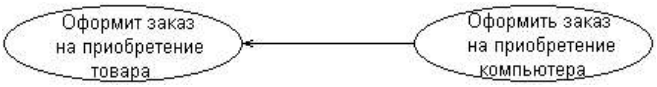

Рис. 7 Пример графического изображения отношения обобщения между вариантами использования

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

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. Данное отношение является направленным бинарным отношением в том смысле, что пара экземпляров вариантов использования всегда упорядочена в отношении включения.

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.

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 (вариант отношения зависимости), направленной от базового варианта использования к включаемому. При этом данная линия со стрелкой помечается ключевым словом "include" ("включает"), как показано на рис. 8

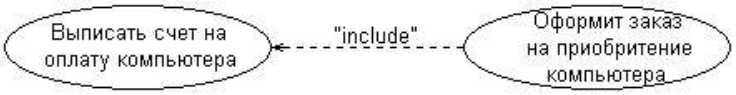

Рис. 8 Пример графического изображения отношения включения между вариантами использования

В языке UML взаимодействие элементов рассматривается в информационном аспекте их коммуникации, т. е. взаимодействующие объекты обмениваются между собой некоторой информацией. При этом информация принимает форму законченных сообщений.

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

Объекты.

На диаграмме последовательности изображаются исключительно те объекты, которые непосредственно участвуют во взаимодействии и не показываются возможные

статические ассоциации с другими объектами. Для диаграммы последовательности ключевым моментом является именно динамика взаимодействия объектов во времени. При этом диаграмма последовательности имеет как бы два измерения. Одно - слева направо в виде вертикальных линий, каждая из которых изображает линию жизни отдельного объекта, участвующего во взаимодействии. Графически каждый объект изображается прямоугольником и располагается в верхней части своей линии жизни (рис. 9). Внутри прямоугольника записываются имя объекта и имя класса, разделенные двоеточием. При этом вся запись подчеркивается, что является признаком объекта.

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

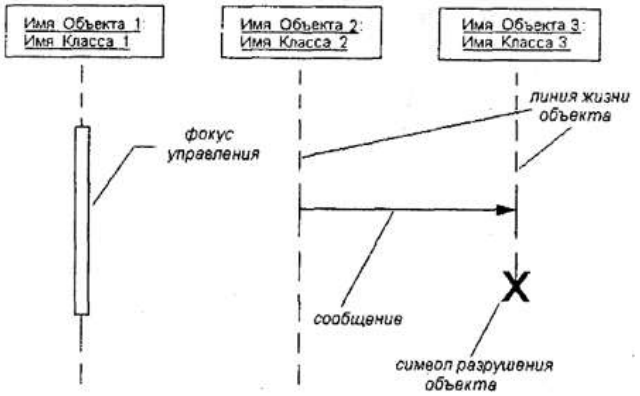

Рис. 9. Различные графические примитивы диаграммы последовательности

Крайним слева на диаграмме изображается объект, который является инициатором взаимодействия (объект 1 на рис. 9). Правее изображается другой объект, который непосредственно взаимодействует с первым. Таким образом, все объекты на диаграмме последовательности образуют некоторый порядок, определяемый степенью активности этих объектов при взаимодействии друг с другом.

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

Линия жизни объекта

Линия жизни объекта (object lifeline) изображается пунктирной вертикальной линией, ассоциированной с единственным объектом на диаграмме последовательности. Линия жизни служит для обозначения периода времени, в течение которого объект существует в системе и, следовательно, может потенциально участвовать во всех ее взаимодействиях. Если объект существует в системе постоянно, то и его линия жизни должна продолжаться по всей плоскости диаграммы последовательности от самой верхней ее части до самой нижней (объекты 1 и 2 на рис. 9).

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены (разрушены), чтобы освободить занимаемые ими ресурсы. Для таких объектов линия жизни обрывается в момент его уничтожения. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы "X" (объект 3 на рис. 9). Ниже этого символа пунктирная линия не изображается, поскольку

соответствующего объекта в системе уже нет, и этот объект должен быть исключен из всех последующих взаимодействий.

Вовсе не обязательно создавать все объекты в начальный момент времени. Отдельные объекты в системе могут создаваться по мере необходимости, существенно экономя ресурсы системы и повышая ее производительность. В этом случае прямоугольник такого объекта изображается не в верхней части диаграммы последовательности, а в той ее части, которая соответствует моменту создания объекта (объект 6 на рис. 10). При этом прямоугольник объекта вертикально располагается в том месте диаграммы, которое по оси времени совпадает с моментом его возникновения в системе. Очевидно, объект обязательно создается со своей линией жизни и, возможно, с фокусом управления.

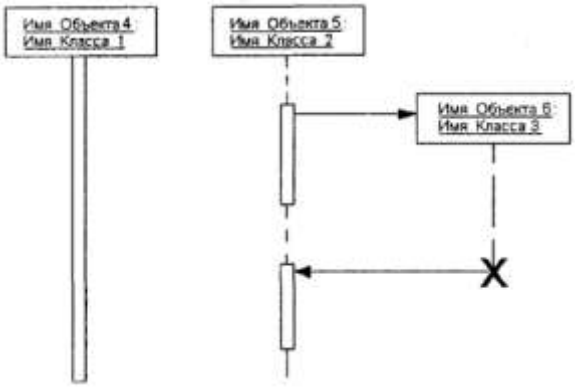

Рис. 10 Графическое изображение различных вариантов линий жизни и фокусов управления объектов

Фокус управления

В процессе функционирования объектно-ориентированных систем одни объекты могут находиться в активном состоянии, непосредственно выполняя определенные действия или в состоянии пассивного ожидания сообщений от других объектов. Чтобы явно выделить подобную активность объектов, в языке UML применяется специальное понятие, получившее название фокуса управления (focus of control). Фокус управления изображается в форме вытянутого узкого прямоугольника (см. объект 1 на рис. 9), верхняя сторона которого обозначает начало получения фокуса управления объекта (начало активности), а ее нижняя сторона - окончание фокуса управления (окончание активности). Этот прямоугольник располагается ниже обозначения соответствующего объекта и может заменять его линию жизни (объект 4 на рис. 10), если на всем ее протяжении он является активным.

С другой стороны, периоды активности объекта могут чередоваться с периодами его пассивности или ожидания. В этом случае у такого объекта имеются несколько фокусов управления (объект 5 на рис. 10). Важно понимать, что получить фокус управления может только существующий объект, у которого в этот момент имеется линия жизни. Если же некоторый объект был уничтожен, то вновь возникнуть в системе он уже не может. Вместо него лишь может быть создан другой экземпляр этого же класса, который, строго говоря, будет являться другим объектом.

В отдельных случаях инициатором взаимодействия в системе может быть актер или внешний пользователь. В этом случае актер изображается на диаграмме последовательности самым первым объектом слева со своим фокусом управления (рис. 11). Чаще всего актер и его фокус управления будут существовать в системе постоянно, отмечая характерную для пользователя активность в инициировании взаимодействий с системой. При этом сам актер может иметь собственное имя либо оставаться анонимным.

Иногда некоторый объект может инициировать рекурсивное взаимодействие с самим собой. Речь идет о том, что наличие во многих языках программирования специальных средств построения рекурсивных процедур требует визуализации соответствующих понятий в форме графических примитивов. На диаграмме последовательности рекурсия обозначается небольшим прямоугольником, присоединенным

к правой стороне фокуса управления того объекта, для которого изображается это рекурсивное взаимодействие (объект 7 на рис. 11).

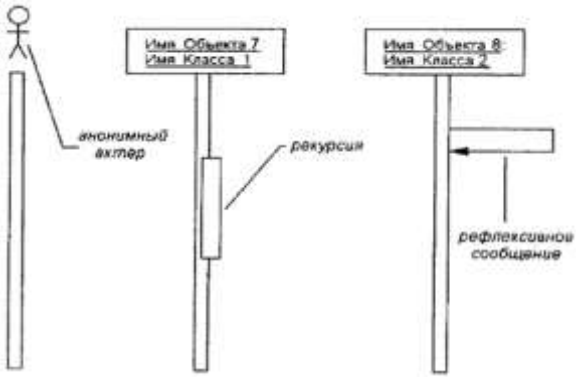

Рис. 11. Графическое изображение актера, рекурсии и рефлексивного сообщения на диаграмме последовательности

Сообщения

Как было отмечено выше, цель взаимодействия в контексте языка UML заключается в том, чтобы специфицировать коммуникацию между множеством взаимодействующих объектов. Каждое взаимодействие описывается совокупностью сообщений, которыми участвующие в нем объекты обмениваются между собой. В этом смысле сообщение (message) представляет собой законченный фрагмент информации, который отправляется одним объектом другому. При этом прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено.

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

В таком контексте каждое сообщение имеет направление от объекта, который инициирует и отправляет сообщение, к объекту, который его получает. Иногда отправителя сообщения называют клиентом, а получателя - сервером. При этом сообщение от клиента имеет форму запроса некоторого сервиса, а реакция сервера на запрос после получения сообщения может быть связана с выполнением определенных действий или передачи клиенту необходимой информации тоже в форме сообщения.

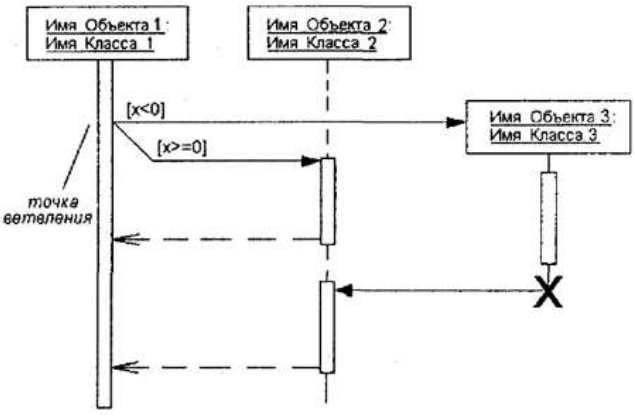

Рис. 12.Графическое изображение различных видов сообщений между объектами на диаграмме последовательности

В языке UML могут встречаться несколько разновидностей сообщений, каждое из которых имеет свое графическое изображение (рис. 12).

- Первая разновидность сообщения является наиболее распространенной и используется для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления. Начало этой стрелки всегда соприкасается с фокусом управления или линией жизни того объекта-клиента, который инициирует это сообщение. Конец стрелки соприкасается с линией жизни того объекта, который принимает это сообщение и выполняет в ответ определенные действия. При этом принимающий объект зачастую получает и фокус управления, становясь активным.

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

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

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

Таким образом, в языке UML каждое сообщение ассоциируется с некоторым действием, которое должно быть выполнено принявшим его объектом. При этом действие может иметь некоторые аргументы или параметры, в зависимости от конкретных значений которых может быть получен различный результат. Соответствующие параметры будет иметь и вызывающее это действие сообщение. Более того, значения параметров отдельных сообщений могут содержать условные выражения, образуя ветвление или альтернативные пути основного потока управления.

Ветвление потока управления

Для изображения ветвления рисуются две или более стрелки, выходящие из одной точки фокуса управления объекта (фокус управления объекта 1 на рис. 13). При этом соответствующие условия должны быть явно указаны рядом с каждой из стрелок в форме сторожевого условия. Как нетрудно представить, если условие записано в форме булевского выражения, то ветвление будет содержать только две ветви. В любом случае условия должны взаимно исключать одновременную передачу альтернативных сообщений.

С помощью ветвления можно изобразить и более сложную логику взаимодействия объектов между собой. Если условий более двух, то для каждого из них необходимо

предусмотреть ситуацию единственного выполнения.

Рис. 13 Графическое изображение бинарного ветвления потока управления на диаграмме последовательности

Стереотипы сообщений

В языке UML предусмотрены некоторые стандартные действия, выполняемые в ответ на получение соответствующего сообщения. Они могут быть явно указаны на диаграмме последовательности в форме стереотипа рядом с сообщением, к которому они относятся. В этом случае они записываются в кавычках. Используются следующие обозначения для моделирования действий:

"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 (перехватывается) другим. Отличие сигнала от сообщения заключается в том, что сигнал должен быть явно описан в том классе, объект которого инициирует его передачу.

Временные ограничения на диаграммах последовательности

В отдельных случаях выполнение тех или иных действий на диаграмме последовательности может потребовать явной спецификации временных ограничений, накладываемых на сам интервал выполнения операций или передачу сообщений. В языке UML для записи временных ограничений используются фигурные скобки. Временные ограничения могут относиться как к выполнению определенных действий объектами, так и к самим сообщениям, явно специфицируя условия их передачи или приема. Важно понимать, что в отличие от условий ветвления, которые должны выполняться альтернативно, временные ограничения имеют обязательный или директивный характер для ассоциированных с ними объектов.

Временные ограничения могут записываться рядом с началом стрелки соответствующего сообщения. Но наиболее часто они записываются слева от этой стрелки

на одном уровне с ней. Если временная характеристика относится к конкретному объекту, то имя этого объекта записывается перед именем характеристики и отделяется от нее точкой.

Примерами таких ограничений на диаграмме последовательности могут служить ситуации, когда необходимо явно специфицировать время, в течение которого допускается передача сообщения от клиента к серверу или обработка запроса клиента сервером:

{время_приема_сообщения время_отправки_сообщения < 1 сек.}

{время_ожидания_ответа < 5 сек.}

{время_передачи_пакета < 10 сек.}

{объект_1. время_подачи_сигнала_тревоги > 30 сек.}

Комментарии или примечания

Комментарии или примечания уже рассматривались ранее при изучении других видов диаграмм. Они могут включаться и в диаграммы последовательности, ассоциируясь с отдельными объектами или сообщениями. При этом используется стандартное обозначение для комментария - прямоугольник с "заломленным" правым верхним углом. Внутри этого прямоугольника записывается текст комментария на естественном языке.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. На основе приведённых примеров рассмотреть основные принципы построения диаграмм вариантов и последовательности на основе языка UML.
3. По выбранному варианту построить диаграмму вариантов использования для соответствующей информационной системы.
4. По выбранному варианту построить диаграмму последовательности для соответствующей информационной системы.
5. В электронном виде оформите краткий отчет о выполненной работе.
6. Сдайте выполненную работу.

Форма представления результата:

Построенные диаграммы, отчет о работе.

Критерии оценки:

Отлично - работа выполнена полностью, без ошибок, сделан и сдан отчет.

Хорошо - работа выполнена полностью, с незначительными ошибками или неточностями, сделан и сдан отчет.

Удовлетворительно - работа выполнена не полностью, со незначительными ошибками или неточностями, отчет не сделан или сделан частично.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 2 Построение диаграммы Кооперации и диаграммы Развертывания и генерация кода

Цель: ознакомиться с методологией моделирования диаграммы Кооперации и диаграммы Развертывания на основе языка UML

Выполнив работу, Вы будете:

уметь:

- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств.
- проектировать и разрабатывать систему по заданным требованиям и спецификациям.
- использовать методологию языка UML для моделирования диаграмм Кооперации и Развертывания

Материальное обеспечение:

Персональный компьютер

Задание:

1. Ознакомиться с теоретическим материалом.
2. Ознакомиться с методологией построения диаграммы Кооперации и диаграммы Развертывания на основе языка UML.
3. Постройте диаграмму Кооперации для выбранной информационной системы
4. Выполните построение диаграммы Развертывания для выбранной информационной системы
5. Сдайте выполненную работу.

Краткие теоретические сведения:

Особенности взаимодействия элементов моделируемой системы могут быть представлены на диаграммах последовательности и кооперации. Диаграмма последовательности служит для визуализации **временных** аспектов взаимодействия, 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.

Прежде всего, 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. Следовательно, если необходимо явно специфицировать взаимосвязи между объектами в реальном времени, лучше это делать на диаграмме последовательности.

Поведение системы может описываться на уровне отдельных объектов, которые обмениваются между собой сообщениями, чтобы достичь нужной цели или реализовать некоторый сервис. С точки зрения аналитика или конструктора важно представить в проекте системы структурные связи отдельных объектов между собой. Такое статическое представление структуры системы как совокупности взаимодействующих объектов и обеспечивает диаграмма ***кооперации.***

Кооперация

Понятие кооперации (collaboration) является одним из фундаментальных понятий в языке UML. Оно служит для обозначения множества взаимодействующих с определенной целью объектов в общем контексте моделируемой системы. Цель самой кооперации состоит в том, чтобы специфицировать особенности реализации отдельных наиболее значимых операций в системе. Кооперация определяет структуру поведения системы в терминах взаимодействия участников этой кооперации.

Кооперация может быть представлена на двух уровнях:

- На уровне спецификации - показывает роли классификаторов и роли ассоциаций в рассматриваемом взаимодействии.

- На уровне примеров - указывает экземпляры и связи, образующие отдельные роли в кооперации.

Диаграмма кооперации ***уровня спецификации*** показывает роли, которые играют участвующие во взаимодействии элементы. Элементами кооперации на этом уровне являются классы и ассоциации, которые обозначают отдельные роли классификаторов и ассоциации между участниками кооперации.

Диаграмма кооперации ***уровня примеров*** представляется совокупностью объектов (экземпляры классов) и связей (экземпляры ассоциаций). При этом связи дополняются стрелками сообщений. На данном уровне показываются только релевантные объекты, т. е. имеющие непосредственное отношение к реализации операции или классификатора.

В кооперации уровня примеров определяются свойства, которые должны иметь экземпляры для того, чтобы участвовать в кооперации. Кроме свойств объектов на диаграмме кооперации также указываются ассоциации, которые должны иметь место между объектами кооперации. При этом вовсе не обязательно изображать все свойства или все ассоциации, поскольку на диаграмме кооперации присутствуют только роли классификаторов, но не сами классификаторы. Таким образом, в то время как классификатор требует полного описания всех своих экземпляров, роль классификатора требует описания только тех свойств и ассоциаций, которые необходимы для участия в отдельной кооперации.

Отсюда вытекает важное следствие. Одна и та же совокупность объектов может участвовать в различных кооперациях. При этом, в зависимости от рассматриваемой кооперации, могут изменяться как свойства отдельных объектов, так и связи между ними. Именно это отличает диаграмму кооперации от диаграммы классов, на которой должны быть указаны все свойства и ассоциации между элементами диаграммы.

Диаграмма кооперации уровня спецификации

Кооперация на уровне спецификации изображается на диаграмме пунктирным эллипсом, внутри которого записывается имя этой кооперации (рис. 1). Такое представление кооперации относится к отдельному варианту использования и детализирует особенности его последующей реализации. Символ эллипса кооперации соединяется отрезками пунктирной линии с каждым из участников этой кооперации, в качестве которых могут выступать объекты или классы. Каждая из этих пунктирных линий помечается ролью (role) участника. Роли соответствуют именам элементов в контексте всей кооперации. Эти имена трактуются как параметры, которые ограничивают спецификацию элементов при любом их появлении в отдельных представлениях модели.

Рис. 1. Общее представление кооперации на диаграммах уровня спецификации

Простой класс на диаграмме кооперации обозначается прямоугольником класса, внутри которого записывается строка текста. Эта строка текста называется ролью классификатора (classifier role). Роль классификатора показывает особенность использования объектов данного класса. Обычно в прямоугольнике показывается только секция имени класса, хотя не исключается возможность указания секций атрибутов и операций.

Строка текста в прямоугольнике должна иметь следующий формат:

'/' <Имя роли классификатора> ':' <Имя классификатора>

[':' <Имя классификатора >]*

Здесь Имя классификатора, если это необходимо, может включать полный путь всех вложенных пакетов. При этом один пакет от другого отделяется двойным двоеточием "::". Если не возникает путаницы, можно ограничиться указанием только ближайшего из пакетов, которому принадлежит данная кооперация. Символ "*" применяется для указания возможности итеративного повторения имени классификатора.

Если кооперация допускает обобщенное представление, то на диаграммах могут быть указаны отношения обобщения соответствующих элементов. Этот способ может быть использован для определения отдельных кооперации, которые являются, в свою очередь, частным случаем или специализацией другой кооперации. Такая ситуация изображается обычной стрелкой обобщения, направленной от символа дочерней кооперации к символу кооперации-предка (рис. 2). При этом роли дочерних кооперации могут быть специализациями ролей кооперации-предков.

Рис. 2.Графическое изображение отношения обобщения между отдельными кооперациями уровня спецификации

В отдельных случаях возникает необходимость явно указать тот факт, что кооперация является реализацией некоторой операции или классификатора. Это можно представить одним из двух способов.

Во-первых, можно соединить символ кооперации пунктирной линией со стрелкой обобщения с символом класса, реализацию операции которого специфицирует данная кооперация (рис. 3, а). Так, если в качестве класса рассмотреть "Заказ на покупку товара", у которого имеется операция "оформить_заказ (), то ее реализация может быть специфицирована в форме кооперации.

Рис. 3. Способы представления кооперации, которая реализует операцию класса

Во-вторых, можно просто изобразить символ кооперации, внутри которого указать всю необходимую информацию, записанную по определенным правилам (рис. 3, б). Эти правила определяют формат записи имени кооперации, после которого записывают двоеточие и имя класса. За именем класса следует двойное двоеточие и имя операции.

Подобное общее представление кооперации на уровне спецификации используется на начальных этапах проектирования. В последующем каждая из кооперацией подлежит детализации на уровне примеров, на котором раскрывается содержание и структура взаимосвязей ее элементов на отдельной диаграмме кооперации. При этом в качестве элементов диаграммы кооперации выступают объекты и связи, дополненные сообщениями.

Объекты

Объекты являются основными элементами или графическими примитивами, из которых строится диаграмма кооперации на уровне примеров. Для графического изображения объектов используется такой же символ прямоугольника, что и для классов.

Объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он может иметь свое собственное имя и конкретные значения атрибутов. Применительно к объектам формат строки классификатора дополняется именем объекта и приобретает следующий вид (при этом вся запись подчеркивается):

<Имя объекта>'/' <Имя роли классификатора> ':' <Имя классификатора>

[':' <Имя классификатора >]*

Здесь Имя роли классификатора может не указываться. В этом случае оно исключается из строки текста вместе с последующим двоеточием. Имя роли может быть опущено в том случае, если существует только одна роль в кооперации, которую могут играть объекты, созданные на базе этого класса.

Таким образом, для обозначения роли классификатора достаточно указать либо имя класса (вместе с двоеточием), либо имя роли (вместе с наклонной чертой). В противном случае прямоугольник будет соответствовать обычному классу. Если роль, которую должен играть объект, наследуется от нескольких классов, то все они должны быть указаны явно и разделяться запятой и двоеточием.

Возможные варианты записи строки текста в прямоугольнике объекта.

: C - анонимный объект, образуемый на основе класса C.

/ R - анонимный объект, играющий роль R.

/ R : C - анонимный объект, образуемый на основе класса C и играющий роль R.

O / R - объект с именем O, играющий роль R.

O : C - объект с именем O, образуемый на основе класса C.

O / R : C - объект с именем O, образуемый на основе класса C и играющий роль R.

O или - объект с именем O.

O : - "объект-сирота" с именем O.

/ R - роль с именем R

: C - анонимная роль на базе класса C.

/ R : C - роль с именем R на основе класса C.

Отдельные примеры изображения объектов и классов на диаграмме кооперации приводятся на следующем рисунке (рис. 4).

Рис. 4. Примеры различных вариантов записи имен объектов, ролей и классов на диаграммах кооперации

На (рис. 4, а) обозначен объект с именем "клиент", играющий роль "инициатор запроса". На (рис. 4, б) показано обозначение анонимного объекта, который играет роль инициатора запроса. В обоих случаях не указан класс, на основе которого будут созданы эти объекты. Обозначение класса присутствует в следующем варианте записи (рис. 4, в), причем объект также анонимный.

Применительно к уровню спецификации на диаграммах кооперации могут присутствовать именованные классы с указанием роли класса в кооперации (рис. 4, г) или анонимные классы, когда указывается только его роль (рис. 4, д). Последний случай характерен для ситуации, когда в модели могут присутствовать несколько классов с именем "Клиент", поэтому требуется явно указать имя соответствующего пакета База данных (рис. 4, е).

Мультиобъект

Мультиобъект (multiobject) представляет собой целое множество объектов на одном из концов ассоциации. На диаграмме кооперации Мультиобъект используется для того, чтобы показать операции и сигналы, которые адресованы всему множеству объектов, а не только одному. Мультиобъект изображается двумя прямоугольниками, один из которых выступает из-за верхней правой вершины другого (рис. 5, а). При этом стрелка сообщения относится ко всему множеству объектов, которые обозначают данный мульти-объект. На диаграмме кооперации может быть явно указано отношение композиции между мультиобъектом и отдельным объектом из его множества (рис. 5, б).

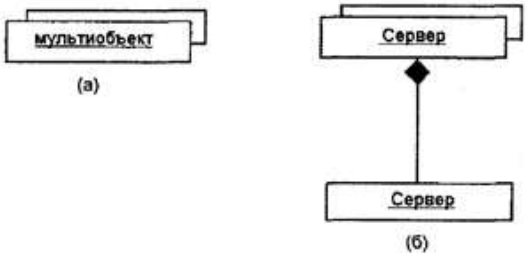

Рис. 5. Графическое изображение мультиобъектов на диаграмме кооперации

Активный объект

В контексте языка UML все объекты делятся на две категории: пассивные и активные. **Пассивный** объект оперирует только данными и не может инициировать деятельность по управлению другими объектами. Однако пассивные объекты могут посылать сигналы в процессе выполнения запросов, которые они получают.

Активный объект (active object) имеет свою собственную нить (thread) управления и может инициировать деятельность по управлению другими объектами. При этом под нитью понимается некоторый облегченный поток управления, который может выполняться параллельно с другими вычислительными нитями или нитями управления в пределах одного вычислительного процесса или процесса управления.

Активные объекты на канонических диаграммах обозначаются прямоугольником с более широкими границами. Иногда может быть явно указано ключевое слово (помеченное значение) {active}, чтобы выделить активный объект на диаграмме. Каждый активный

объект может инициировать единственную нить или процесс управления и представлять исходную точку потока управления.

Составной объект

Составной объект (composite object) или объект-контейнер предназначен для представления объекта, имеющего собственную структуру и внутренние потоки (нити) управления. Составной объект является экземпляром составного класса (класса-контейнера), который связан отношением агрегации или композиции со своими частями. Аналогичные отношения связывают между собой и соответствующие объекты.

На диаграммах кооперации такой составной объект изображается как обычный объект, состоящий из двух секций: верхней и нижней. В верхней секции записывается имя составного объекта, а в нижней - его составные части вместо списка его атрибутов (рис. 6). При этом допускается иметь в качестве частей другие составные объекты.

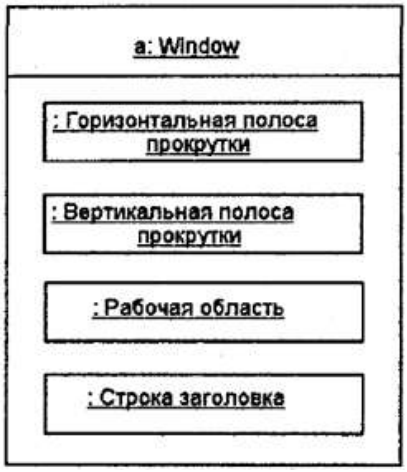

Рис. 6. Графическое изображение составного объекта на диаграмме кооперации

Связи

Связь (link) является экземпляром или примером произвольной ассоциации. Связь как элемент языка UML может иметь место между двумя и более объектами. Бинарная связь на диаграмме кооперации изображается отрезком прямой линии, соединяющей два прямоугольника объектов. На каждом из концов этой линии могут быть явно указаны имена ролей данной ассоциации. Рядом с линией в ее средней части может записываться имя соответствующей ассоциации.

Связи не имеют собственных имен, поскольку полностью идентичны как экземпляры ассоциации. Другими словами, все связи на диаграмме кооперации могут быть только анонимными и записываются без двоеточия перед-именем ассоциации. Для связей не указывается также и кратность. Однако другие обозначения специальных случаев ассоциации (агрегация, композиция) могут присутствовать на отдельных концах связей.

Стереотипы связей

Связь может иметь некоторые стереотипы, которые записываются рядом с одним из ее концов и указывают на особенность реализации данной связи. В языке UML для этой цели могут использоваться следующие стереотипы:

"association" - ассоциация (предполагается по умолчанию, поэтому этот стереотип можно не указывать).

"parameter" - параметр метода. Соответствующий объект может бытьч только параметром некоторого метода.

"local" - локальная переменная метода. Ее область видимости ограничена только соседним объектом.

"global" - глобальная переменная. Ее область видимости распространяется на всю диаграмму кооперации.

"self - рефлексивная связь объекта с самим собой, которая допускает передачу объектом сообщения самому себе. На диаграмме кооперации рефлексивная связь изображается петлей в верхней части прямоугольника объекта.

Сообщения

При построении диаграммы кооперации сообщения имеют некоторые дополнительные семантические особенности. Сообщение на диаграмме кооперации специфицирует коммуникацию между двумя объектами, один из которых передает другому некоторую информацию. При этом первый объект ожидает, что после получения сообщения вторым объектом последует выполнение некоторого действия. Таким образом, именно сообщение является причиной или стимулом для начала выполнения операций, отправки сигналов, создания и уничтожения отдельных объектов. Связь обеспечивает канал для направленной передачи сообщений между объектами от объекта-источника к объекту-получателю.

Рис. 7. Графическое изображение различных типов сообщений на диаграмме кооперации

Сообщения в языке UML также специфицируют роли, которые играют объекты - отправитель и получатель сообщения. Сообщения на диаграмме кооперации изображаются помеченными стрелками рядом (выше или ниже) с соответствующей связью или ролью ассоциации. Направление стрелки указывает на получателя сообщения. Внешний вид стрелки сообщения имеет определенный смысл. На диаграммах кооперации может использоваться один из четырех типов стрелок для обозначения сообщений (рис. 7):

Сплошная линия с треугольной стрелкой (рис. 7, а) обозначает вызов процедуры или другого вложенного потока управления. Может быть также использована совместно с параллельно активными объектами, когда один из них передает сигнал и ожидает, пока не закончится некоторая вложенная последовательность действий. Обычно все такие сообщения являются синхронными, т. е. инициируемыми по завершении некоторой деятельности или при выполнении некоторого условия.

Сплошная линия с V-образной стрелкой (рис. 7, б) обозначает простой поток управления. Каждая такая стрелка изображает один этап в последовательности потока управления. Обычно все такие сообщения являются асинхронными.

Сплошная линия с полустрелкой (рис. 7, в) используется для обозначения асинхронного потока управления. Соответствующие сообщения формируются в произвольные, заранее не известные моменты времени, как правило, активными объектами. Обычно сообщения этого типа являются начальными в последовательности потока управления и чаще всего инициируются актерами.

Пунктирная линия с V-образной стрелкой (рис. 7, г) обозначает возврат из вызова процедуры. Стрелки этого типа зачастую отсутствуют на диаграммах кооперации, поскольку неявно предполагается их существование после окончания процесса активизации некоторой деятельности.

Пример построения диаграммы кооперации

В качестве примера рассмотрим построение диаграммы кооперации для моделирования процесса телефонного разговора с использованием обычной телефонной сети Объектами в этом примере являются два абонента a и b, два телефонных аппарата c и d, коммутатор и сам разговор как объект моделирования. При этом как коммутатор, так и разговор являются анонимными объектами.

На начальном этапе изобразим все объекты и связи между ними на диаграмме кооперации при помощи соответствующих обозначений (рис. 8). Заметим, что первый телефонный аппарат изображен как активный объект, а второй - как пассивный.

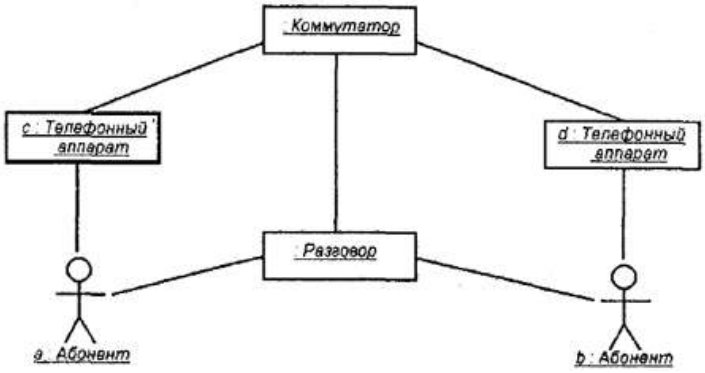

Рис. 8. Начальный фрагмент диаграммы кооперации для примера моделирования обычного телефонного разговора

В последующем необходимо специфицировать все связи на этой диаграмме, указав на их концах необходимую информацию в форме ролей связей. Дополненный таким образом вариант диаграммы кооперации изображен ниже (рис. 9). Заметим, что для объекта "Разговор" указано помеченное значение {transient}, которое означает, что этот объект создается в процессе выполнения объемлющего процесса и уничтожается до его завершения. Напомним, что помеченные значения (tagged values) являются стандартными элементами языка UML.

Рис. 9. фрагмент диаграммы кооперации, дополненный стереотипами ролей связей, именами ассоциаций и помеченным значением объекта.

Наконец, на диаграмму кооперации необходимо нанести все сообщения, указав их порядок и семантические особенности. Окончательный фрагмент диаграммы кооперации изображен на рис. 10 и содержит, строго говоря, модель кооперации только для начала разговора. Эта диаграмма может быть дополнена сообщениями, необходимыми для окончания разговора, что читателям предлагается выполнить самостоятельно в качестве упражнения.

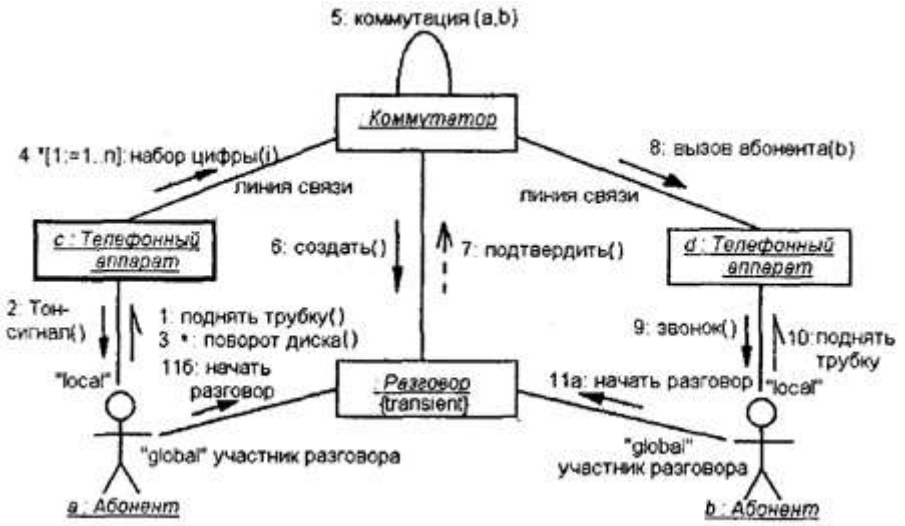

Рис. 10. Окончательный вариант диаграммы кооперации для моделирования телефонного разговора

Если взаимодействующие объекты образуют между собой различные типы отношений-ассоциаций (композиция, агрегация), то диаграмма кооперации оказывается необходимым представлением модели на всех ее уровнях.

Диаграмма Развертывания.

Физическое представление программной системы не может быть полным, если отсутствует информация о том, на какой платформе и на каких вычислительных средствах она реализована. Конечно, если разрабатывается простая программа, которая может выполняться локально на компьютере пользователя, не задействуя никаких периферийных устройств и ресурсов, то в этом случае нет необходимости в разработке дополнительных диаграмм.

В случае разработки сложных программных средств существует ряд особенностей:

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

- интеграция программной системы с Интернетом определяет необходимость решения дополнительных вопросов при проектировании системы, таких как обеспечение безопасности, криптозащищенности и устойчивости доступа к информации для корпоративных клиентов. Эти аспекты в немалой степени зависят от реализации проекта в форме физически существующих узлов системы, таких как серверы, рабочие станции, брандмауэры, каналы связи и хранилища данных.

- технологии доступа и манипулирования данными в рамках общей схемы "клиент-сервер" также требуют размещения больших баз данных в различных сегментах корпоративной сети, их резервного копирования, архивирования, кэширования для обеспечения необходимой производительности системы в целом. Эти аспекты также требуют визуального представления с целью спецификации программных и технологических особенностей реализации распределенных архитектур.

Первой из диаграмм физического представления является диаграмма компонентов. Второй формой физического представления программной системы является диаграмма **развертывания** (синоним - диаграмма размещения). Она применяется для представления общей конфигурации и топологии распределенной программной системы и содержит распределение компонентов по отдельным узлам системы. Кроме того, диаграмма развертывания показывает наличие физических соединений - маршрутов передачи информации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развертывания предназначена для визуализации элементов и компонентов программы, существующих лишь на этапе ее исполнения (runtime). При этом представляются только компоненты-экземпляры программы, являющиеся исполнимыми файлами или динамическими библиотеками. Те компоненты, которые не используются на этапе исполнения, на диаграмме развертывания не показываются. Так, компоненты с исходными текстами программ могут присутствовать только на диаграмме компонентов. На диаграмме развертывания они не указываются.

Диаграмма развертывания содержит графические изображения процессоров, устройств, процессов и связей между ними. В отличие от диаграмм логического представления, диаграмма развертывания является единой для системы в целом, поскольку должна всецело отражать особенности ее реализации. Эта диаграмма, по сути, завершает процесс ООАП для конкретной программной системы и ее разработка, как правило, является последним этапом спецификации модели.

Итак, перечислим цели, преследуемые при разработке диаграммы развертывания:

1. Определить распределение компонентов системы по ее физическим узлам.
2. Показать физические связи между всеми узлами реализации системы на этапе ее исполнения.
3. Выявить узкие места системы и реконфигурировать ее топологию для достижения требуемой производительности.

Для обеспечения этих требований диаграмма развертывания разрабатывается совместно системными аналитиками, сетевыми инженерами и системотехниками. Далее рассмотрим отдельные элементы, из которых состоят диаграммы развертывания.

Узел

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

Графически на диаграмме развертывания узел изображается в форме трехмерного куба. Узел имеет собственное имя, которое указывается внутри этого графического символа. Сами узлы могут представляться как в качестве типов (рис. 1, а), так и в качестве экземпляров (рис. 1, б).

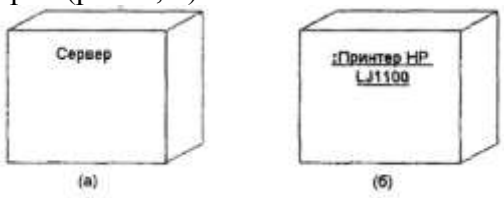

Рис.1. Графическое изображение узла на диаграмме развертывания

В первом случае имя узла записывается без подчеркивания и начинается с заглавной буквы. Во втором имя узла-экземпляра записывается в виде <имя узла ':' имя типа узла>. Имя типа узла указывает на некоторую разновидность узлов, присутствующих в модели системы.

Например, аппаратная часть системы может состоять из нескольких персональных компьютеров, каждый из которых соответствует отдельному узлу-экземпляру в модели. Однако все эти узлы-экземпляры относятся к одному типу узлов, а именно узлу с именем типа "Персональный компьютер". Так, на представленном выше рисунке (рис. 1, а) узел с именем "Сервер" относится к общему типу и никак не конкретизируется. Второй же узел (рис. 1, б) является анонимным узлом-экземпляром конкретной модели принтера.

Так же, как и на диаграмме компонентов, изображения узлов могут расширяться, чтобы включить некоторую дополнительную информацию о спецификации узла. Если

дополнительная информация относится к имени узла, то она записывается под этим именем в форме помеченного значения.

Если необходимо явно указать компоненты, которые размещаются на отдельном узле, то это можно сделать двумя способами. Первый из них позволяет разделить графический символ узла на две секции горизонтальной линией. В верхней секции записывают имя узла, а в нижней секции - размещенные на этом узле компоненты (рис. 2, а).

Второй способ разрешает показывать на диаграмме развертывания узлы с вложенными изображениями компонентов (рис. 2, б). Важно помнить, что в качестве таких вложенных компонентов могут выступать только исполняемые компоненты.

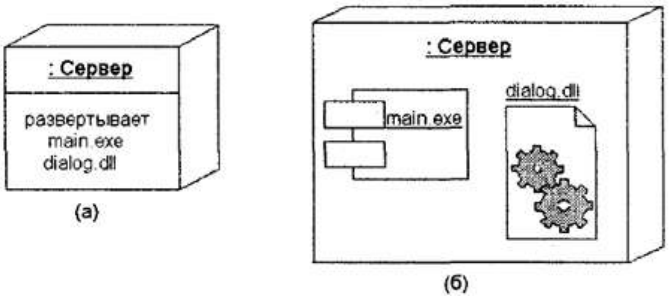

Рис. 2. Варианты графического изображения узлов-экземпляров с размещаемыми на них компонентами

В качестве дополнения к имени узла могут использоваться различные стереотипы, которые явно специфицируют назначение этого узла. Хотя в языке UML стереотипы для узлов не определены, в литературе встречаются следующие их варианты: "процессор", "датчик", "модем", "сеть", "консоль" и др., которые самостоятельно могут быть определены разработчиком. Более того, на диаграммах развертывания допускаются специальные обозначения для различных физических устройств, графическое изображение которых проясняет назначение или выполняемые устройством функции.

Соединения

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

Соединения являются разновидностью ассоциации и изображаются отрезками линий без стрелок. Наличие такой линии указывает на необходимость организации физического канала для обмена информацией между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, помеченным значением или ограничением. Так, на представленном ниже фрагменте диаграммы развертывания (рис. 3) явно определены не только требования к скорости передачи данных в локальной сети с помощью помеченного значения, но и рекомендации по технологии физической реализации соединений в форме примечания.

Рис. 3. Фрагмент диаграммы развертывания с соединениями между узлами

Кроме соединений на диаграмме развертывания могут присутствовать отношения зависимости между узлом и развернутыми на нем компонентами. Подобный способ является альтернативой вложенному изображению компонентов внутри символа узла, что не всегда удобно, поскольку делает этот символ излишне объемным. Поэтому при большом

количестве развернутых на узле компонентов соответствующую информацию можно представить в форме отношения зависимости (рис. 4).

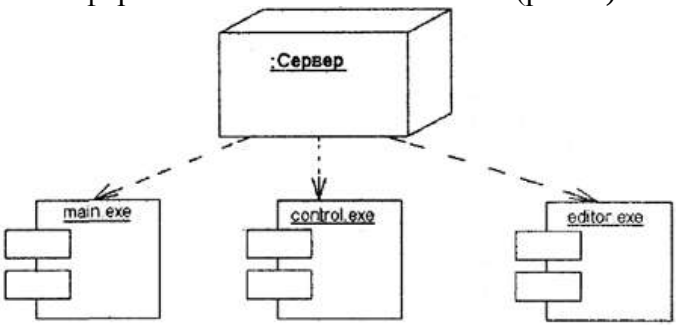

Рис. 4. Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами.

Диаграммы развертывания могут иметь более сложную структуру, включающую вложенные компоненты, интерфейсы и другие аппаратные устройства. На изображенной ниже диаграмме развертывания (рис. 5) представлен фрагмент физического представления системы удаленного обслуживания клиентов банка. Узлами этой системы являются удаленный терминал (узел-тип) и сервер банка (узел-экземпляр).

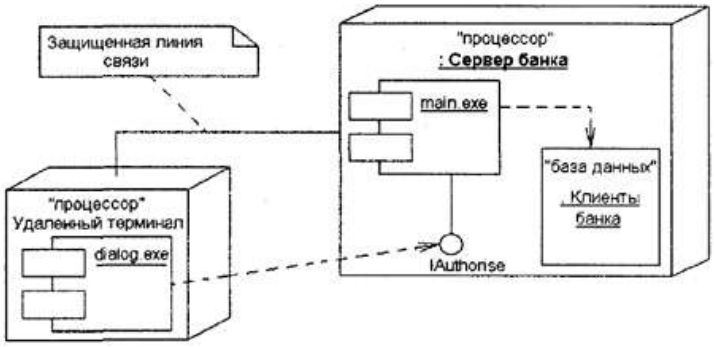

Рис. 5. Диаграмма развертывания для системы удаленного обслуживания клиентов банка

На этой диаграмме развертывания указана зависимость компонента реализации диалога "dialog.exe" на удаленном терминале от интерфейса lAuthorise, реализованного компонентом "main.exe", который, в свою очередь, развернут на анонимном узле-экземпляре "Сервер банка". Последний зависит от компонента базы данных "Клиенты банка", который развернут на этом же узле.

Примечание указывает на необходимость использования защищенной линии связи для обмена данными в данной системе. Другой вариант записи этой информации заключается в дополнении диаграммы узлом со стереотипом "закрытая сеть".

Разработка так называемых встроенных систем предполагает не только создание программного кода, но и согласование между собой всех аппаратных средств и механических устройств. В качестве примера рассмотрим фрагмент модели управления удаленным механическим средством типа транспортной платформы. Такая платформа предназначена для перемещения в агрессивных средах, где присутствие человека невозможно в силу целого ряда физических причин.

Транспортная платформа оснащается собственным микропроцессором, цифровой видеокамерой, датчиками температуры и местоположения, а также управляющими приводами для изменения направления и скорости перемещения платформы. Управляющая и телеметрическая информация от платформы по радиолинии передается в центр управления, оснащенный управляющим компьютером, манипуляторами управления и большим информационным табло.

На микропроцессоре платформы развернуты программные компоненты для реализации простейших управляющих воздействий на приводы, что позволяет дискретно

изменять направление и скорость перемещения платформы. На компьютере центра управления развернуты программные компоненты анализа телеметрической информации, характеризующей состояние отдельных устройств' платформы, а также реализованы алгоритмы управления перемещением платформы в целом.

Вариант физического представления этой транспортной системы показан на следующей диаграмме развертывания (рис. 6)

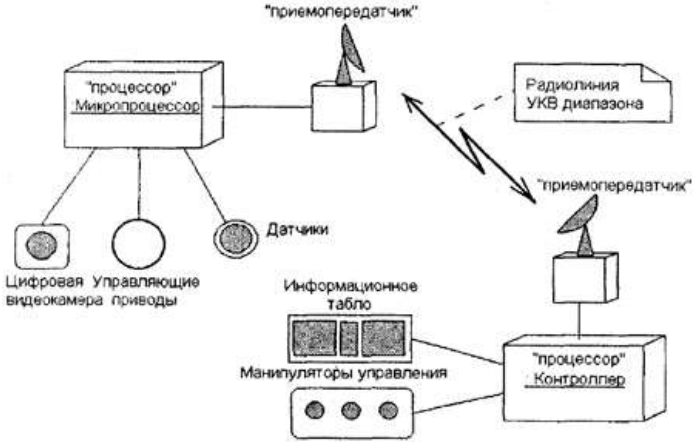

Рис. 6. Диаграмма развертывания для модели системы управления транспортной платформой

Данная диаграмма содержит самую общую информацию о развертывании рассматриваемой системы и в последующем может быть детализирована при разработке собственно программных компонентов управления. Как видно из рисунка, при разработке этой диаграммы развертывания использованы дополнительный стереотип "приемопередатчик", который отсутствует в описании языка UML, и специальные изображения для отдельных аппаратных и механических устройств.

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

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

При разработке простых программ, которые исполняются локально на одном компьютере, так же как и в случае диаграммы компонентов, необходимость в диаграмме развертывания может вообще отсутствовать.

Как правило, разработка диаграммы развертывания осуществляется на завершающем этапе ООАП, что характеризует окончание фазы проектирования физического представления. С другой стороны, диаграмма развертывания может строиться для анализа существующей системы с целью ее последующего анализа и модификации. При этом анализ предполагает разработку этой диаграммы на его начальных этапах, что характеризует общее направление анализа от физического представления к логическому.

При моделировании бизнес-процессов диаграмма развертывания, кроме компьютеров корпоративной сети, может содержать в качестве узлов различные средства оргтехники (факсимильные устройства, многоканальные телефонные станции, множительные аппараты, экраны для презентаций и др.). При этом каждое из подобных устройств может функционировать как автономно, так и в составе корпоративной сети.

Если необходимо включить в модель ресурсы Интернета, то на диаграмме развертывания Интернет обозначается в форме "облачка" с соответствующим именем. Строго говоря, подобное обозначение не специфицировано в языке UML, однако оно часто используется при разработке моделей распределенных систем.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. На основе приведённых примеров рассмотреть основные принципы построения диаграмм кооперации и развертывания на основе языка UML.
3. По выбранному варианту построить диаграмму кооперации для соответствующей информационной системы.
4. По выбранному варианту построить диаграмму развертывания для соответствующей информационной системы.
5. В электронном виде оформите краткий отчет о выполненной работе.
6. Сдайте выполненную работу.

Форма представления результата:

Построенные диаграммы, отчет о работе.

Критерии оценки:

Отлично - работа выполнена полностью, без ошибок, сделан и сдан отчет.

Хорошо - работа выполнена полностью, с незначительными ошибками или неточностями, сделан и сдан отчет.

Удовлетворительно - работа выполнена не полностью, со незначительными ошибками или неточностями, отчет не сделан или сделан частично.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 3 Построение диаграммы Деятельности, диаграммы Состояний и диаграммы Классов и генерация кода

Цель: ознакомиться с методологией моделирования диаграммы Деятельности и диаграммы Состояний и диаграммы Классов на основе языка UML

Выполнив работу, Вы будете:

уметь:

- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств.
- проектировать и разрабатывать систему по заданным требованиям и спецификациям.
- использовать методологию языка UML для моделирования диаграмм Деятельности, Состояний и Классов.

Материальное обеспечение:

Персональный компьютер

Задание:

1. Ознакомиться с теоретическим материалом.
2. Ознакомиться с методологией построения диаграммы Деятельности, диаграммы Состояний и диаграммы Классов на основе языка UML.
3. Выполните построения диаграмм: Деятельности, Состояний, Классов для выбранной информационной системы.
4. Сдайте выполненную работу.

Краткие теоретические сведения:

Моделирование Диаграммы классов

Центральное место в ООАП занимает разработка логической модели системы в виде диаграммы классов.

Нотация UML предоставляет широкие возможности для отображения дополнительной информации (абстрактные операции и классы, стереотипы, общие и частные методы, детализированные интерфейсы, параметризованные классы). При этом возможно использование графических изображений для ассоциаций и их специфических свойств, таких как отношение агрегации, когда составными частями класса могут выступать другие классы.

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о временных аспектах функционирования системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы.

Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких как классы, интерфейсы и отношения между ними и их составляющими компонентами.

В общем случае пакет статической структурной модели может быть представлен в виде одной или нескольких диаграмм классов. Декомпозиция некоторого представления на

отдельные диаграммы выполняется с целью удобства и графической визуализации структурных взаимосвязей предметной области. При этом компоненты диаграммы соответствуют элементам статической семантической модели. Модель системы, в свою очередь, должна быть согласована с внутренней структурой классов, которая описывается на языке UML.

Класс

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. 1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

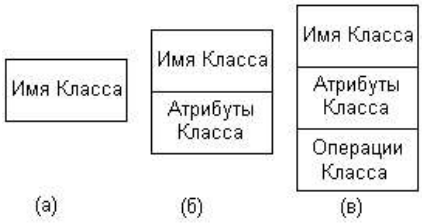

Рис.1. Графическое изображение класса на диаграмме классов

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (рис. 1, а). По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами (рис. 1, б) и операциями (рис. 1, в).

Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится семантическая информация справочного характера или явно указываются исключительные ситуации.

Даже если секция атрибутов и операций является пустой, в обозначении класса она выделяется горизонтальной линией, чтобы сразу отличить класс от других элементов языка UML. Примеры графического изображения классов на диаграмме классов приведены на рис. 2. В первом случае для класса "Прямоугольник" (рис.2, а) указаны только его атрибуты - точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис. 2, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса "Счет" (рис. 2, в) дополнительно изображена четвертая секция, в которой указано исключение - отказ от обработки просроченной кредитной карточки.

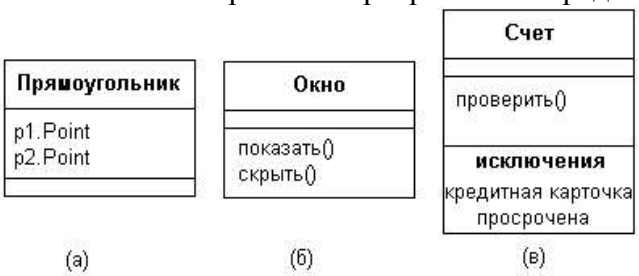

Рис.2. Примеры графического изображения классов на диаграмме

Имя класса

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

имен классов использовать существительные, записанные по практическим соображениям без пробелов.

В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и, соответственно, от которых он наследует свойства и методы. В этой секции может приводиться информация о разработчике данного класса и статус состояния разработки, а также могут записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.

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

Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом. Данное обстоятельство является семантическим аспектом описания соответствующих элементов языка UML.

Атрибуты класса

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, его кратности, типа значений атрибута и, возможно, его исходного значения:

<квантор видимости><имя атрибута>[кратность]:

<тип атрибута> = <исходное значение>{строка-свойство}

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

- 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.
- И, наконец, знак "-" 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. Эта ситуация отличается от принятых по умолчанию соглашений в традиционных языках программирования, когда отсутствие квантора видимости трактуется как public или private. Однако вместо условных графических обозначений можно записывать соответствующее ключевое слово: public, protected, private.

Операция

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. Запись операций класса в языке UML также стандартизована и подчиняется определенным синтаксическим правилам. При этом каждой операции класса соответствует отдельная строка, которая состоит из квантора видимости операции, имени операции, выражения типа возвращаемого операцией значения и, возможно, строка-свойство данной операции:

<квантор видимости><имя операции>(список параметров):

<выражение типа возвращаемого значения>{строка-свойство}

Квантор видимости, как и в случае атрибутов класса, может принимать одно из трех возможных значений и, соответственно, отображается при помощи специального символа. Символ "+" обозначает операцию с областью видимости типа общедоступный (public). Символ "#" обозначает операцию с областью видимости типа защищенный (protected). И, наконец, символ "-" используется для обозначения операции с областью видимости типа закрытый (private).

Квантор видимости для операции может быть опущен. В этом случае его отсутствие просто означает, что видимость операции не указывается. Вместо условных графических обозначений также можно записывать соответствующее ключевое слово: public, protected, private.

Имя операции представляет собой строку текста, которая используется в качестве идентификатора соответствующей операции и поэтому должна быть уникальной в пределах данного класса. Имя атрибута является единственным обязательным элементом синтаксического обозначения операции.

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

<вид параметра><имя параметра>:<выражение типа>=<значение параметра по умолчанию>.

Здесь вид параметра - есть одно из ключевых слов in, out или inout со значением in по умолчанию, в случае если вид параметра не указывается. Имя параметра есть идентификатор соответствующего формального параметра. Выражение типа является зависимой от конкретного языка программирования спецификацией типа возвращаемого значения для соответствующего формального параметра. Наконец, значение по умолчанию в общем случае представляет собой выражение для значения формального параметра, синтаксис которого зависит от конкретного языка программирования и подчиняется принятым в нем ограничениям.

Выражение типа возвращаемого значения также является зависимой от языка реализации спецификацией типа или типов значений параметров, которые возвращаются объектом после выполнения соответствующей операции. Двоеточие и выражение типа возвращаемого значения могут быть опущены, если операция не возвращает никакого значения. Для указания кратности возвращаемого значения данная спецификация может быть записана в виде списка отдельных выражений.

Строка-свойство служит для указания значений свойств, которые могут быть применены к данному элементу. Строка-свойство не является обязательной, она может отсутствовать, если никакие свойства не специфицированы.

Операция с областью действия на весь класс показывается подчеркиванием имени и строки выражения типа. По умолчанию под областью операции понимается объект класса. В этом случае имя и строка выражения типа операции не подчеркиваются.

Операция, которая не может изменять состояние системы и, соответственно, не имеет никакого побочного эффекта, обозначается строкой-свойством "{запрос}" ("{query}"). В противном случае операция может изменять состояние системы, хотя нет никаких гарантий, что она будет это делать.

Для повышения производительности системы одни операции могут выполняться параллельно или одновременно, а другие - только последовательно. В этом случае для указания параллельности выполнения операции используется строка-свойство вида "{concurrency = имя}", где имя может принимать одно из следующих значений: последовательная (sequential), параллельная (concurrent), охраняемая (guarded). При этом придерживаются следующей семантики для данных значений:

- последовательная (sequential) - для данной операции необходимо обеспечить ее единственное выполнение в системе, одновременное выполнение других операций может привести к ошибкам или нарушениям целостности объектов класса.
- параллельная (concurrent) - данная операция в силу своих особенностей может выполняться параллельно с другими операциями в системе, при этом параллельность должна поддерживаться на уровне реализации модели.
- охраняемая (guarded) - все обращения к данной операции должны быть строго упорядочены во времени с целью сохранения целостности объектов данного класса, при этом могут быть приняты дополнительные меры по контролю исключительных ситуаций на этапе ее выполнения.

Отношения между классами

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

1. Отношение зависимости (dependency relationship)
2. Отношение ассоциации (association relationship)
3. Отношение обобщения (generalization relationship)
4. Отношение реализации (realization relationship)

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

1. Отношение зависимости

Отношение зависимости в общем случае указывает некоторое семантическое отношение между двумя элементами модели или двумя множествами таких элементов, которое не является отношением ассоциации, обобщения или реализации. Оно касается только самих элементов модели и не требует множества отдельных примеров для пояснения своего смысла. Отношение зависимости используется в такой ситуации, когда некоторое изменение одного элемента модели может потребовать изменения другого зависимого от него элемента модели.

Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой на одном из ее концов ("->" или "<-"). На диаграмме классов данное отношение связывает отдельные классы между собой, при этом стрелка направлена от класса-клиента зависимости к независимому классу или классу-источнику (рис. 5.3). На данном рисунке изображены два класса: Класс_А и Кяасс_Б, при этом Класс_Б является источником некоторой зависимости, а Класс_А - клиентом этой зависимости.

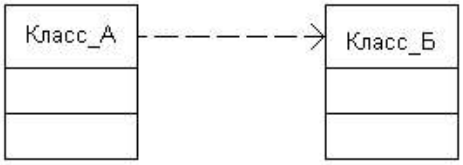

Рис. 3. Графическое изображение отношения зависимости на диаграмме классов

В качестве класса-клиента и класса-источника зависимости могут выступать целые множества элементов модели. В этом случае одна линия со стрелкой, выходящая от источника зависимости, расщепляется в некоторой точке на несколько отдельных линий, каждая из которых имеет отдельную стрелку для класса-клиента. Например, если функционирование Класса_С зависит от особенностей реализации Класса_А и Класса_/>, то данная зависимость может быть изображена следующим образом (рис. 4).

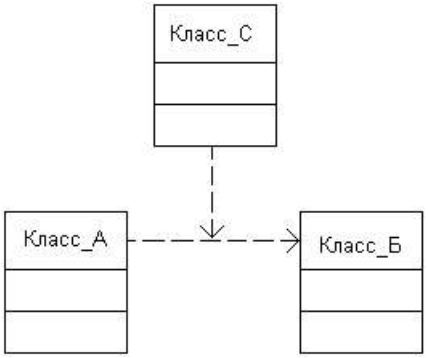

Рис. 4. Графическое представление зависимости между классом-клиентом (Класс_С) и классами-источниками (Класс_Л и Класс_Б)

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

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

2. Отношение ассоциации

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

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

В качестве простого примера отношения бинарной ассоциации рассмотрим отношение между двумя классами - классом "Компания" и классом "Сотрудник" (рис. 5). Они связаны между собой бинарной ассоциацией Работа, имя которой указано на рисунке рядом с линией ассоциации. Для данного отношения определен порядок следования классов, первым из которых является класс "Сотрудник", а вторым - класс "Компания". Отдельным примером или экземпляром данного отношения может являться пара значений (Петров И. И.,

"Рога&Копыта"). Это означает, что сотрудник Петров И. И. работает в компании "Рога&Копыта".

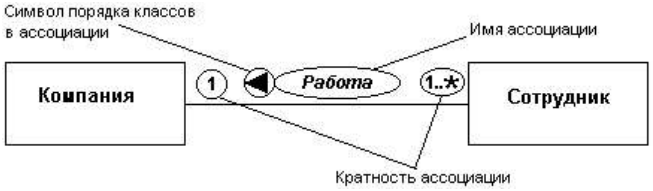

Рис. 5. Графическое изображение отношения бинарной ассоциации между классами

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

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

3. Отношение агрегации

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.

Графически отношение агрегации изображается сплошной линией, один из концов которой представляет собой незакрашенный внутри ромб. Этот ромб указывает на тот из классов, который представляет собой "целое". Остальные классы являются его "частями" (рис. 6).

Рис. 6 Графическое изображение отношения агрегации в языке UML

4. Отношение композиции

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.

Пример - окно интерфейса программы, которое может состоять из строки заголовка, кнопок управления размером, полос прокрутки, главного меню, рабочей области и строки состояния. Графически отношение композиции изображается сплошной линией, один из концов которой представляет собой закрашенный внутри ромб. Этот ромб указывает на тот

из классов, который представляет собой класс-композицию или "целое". Остальные классы являются его "частями" (рис. 7).

Рис. 7. Графическое изображение отношения композиции в языке UML

В качестве дополнительных обозначений для отношений композиции и агрегации могут использоваться дополнительные обозначения, применяемые для отношения ассоциации. А именно, указание кратности класса ассоциации и имени данной ассоциации, которые не являются обязательными. Применительно к описанному выше примеру класса "Окно_программы" его диаграмма классов может иметь следующий вид (рис. 8).

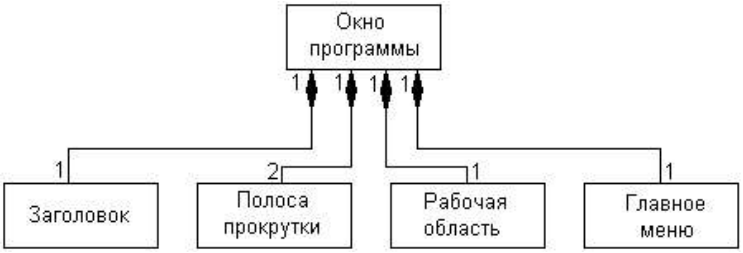

Рис. 8. Диаграмма классов для иллюстрации отношения композиции на примере класса окна программы

5. Отношение обобщения

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

Применительно к диаграмме классов данное отношение описывает иерархическое строение классов и наследование их свойств и поведения. При этом предполагается, что класс-потомок обладает всеми свойствами и поведением класса-предка, а также имеет свои собственные свойства и поведение, которые отсутствуют у класса-предка. На диаграммах отношение обобщения обозначается сплошной линией с треугольной стрелкой на одном из концов (рис. 9). Стрелка указывает на более общий класс (класс-предок или суперкласс), а ее отсутствие - на более специальный класс (класс-потомок или подкласс).

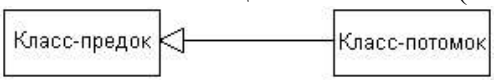

Рис. 9. Графическое изображение отношения обобщения в языке UML

Как правило, на диаграмме может указываться несколько линий для одного отношения обобщения, что отражает его таксономический характер. В этом случае более общий класс разбивается на подклассы одним отношением Обобщения.

Это обозначение по форме соответствует графу специального вида - иерархическому дереву. В этом случае класс-предок является корнем этого дерева, а классы-потомки - его листьями. Отличие заключается в возможности указания на диаграмме классов потенциальной возможности наличия других классов-потомков, которые не включены в обозначения представленных на диаграмме классов (многоточие вместо прямоугольника).

Рядом со стрелкой обобщения может размещаться строка текста, указывающая на некоторые дополнительные свойства этого отношения.

Интерфейсы .

Интерфейсы являются элементами диаграммы вариантов использования. Однако при построении диаграммы классов отдельные интерфейсы могут уточняться и в этом случае для их изображения используется специальный графический символ - прямоугольник класса с ключевым словом или стереотипом "interface" (рис. 10). При этом секция атрибутов у прямоугольника отсутствует, а указывается только секция операций.

Рис. 10. Пример графического изображения интерфейса на диаграмме классов

Объекты

Объект (object) является отдельным экземпляром класса, который создается на этапе выполнения программы. Он имеет свое собственное имя и конкретные значения атрибутов. В силу самых различных причин может возникнуть необходимость показать взаимосвязи не только между классами модели, но и между отдельными объектами, реализующими эти классы. В данном случае может быть разработана диаграмма объектов, которая, хотя и не является канонической в метамодели языка UML, но имеет самостоятельное назначение.

Для графического изображения объектов используется такой же символ прямоугольника, что и для классов. Отличия проявляются при указании имен объектов, которые в случае объектов обязательно подчеркиваются (рис. 11). При этом запись имени объекта представляет собой строку текста "имя объекта:имя класса", разделенную двоеточием (рис. 11 а, б). Имя объекта может отсутствовать, в этом случае предполагается, что объект является анонимным, и двоеточие указывает на данное обстоятельство (рис. 11, г). Отсутствовать может и имя класса. Тогда указывается просто имя объекта (рис. 11, в). Атрибуты объектов принимают конкретные значения.

Рис. 11. Пример графического изображения объектов на диаграммах языка UML

Шаблоны или параметризованные классы

Шаблон (template) или параметризованный класс (parametrized class) предназначен для обозначения такого класса, который имеет один (или более) нефиксированный формальный параметр. Он определяет целое семейство или множество классов, каждый из которых может быть получен связыванием этих параметров с действительными значениями. Обычно параметрами шаблонов служат типы атрибутов классов, такие как целые числа, перечисление, массив строк и др. В более сложном случае формальные параметры могут представлять и операции класса.

Графически шаблон изображается прямоугольником, к верхнему правому углу которого присоединен маленький прямоугольник из пунктирных линий (рис. 12), большой прямоугольник может быть разделен на секции, аналогично обозначению для класса. В верхнем прямоугольнике указывается список формальных параметров для тех классов, которые могут быть получены на основе данного шаблона. В верхней секции шаблона записывается его имя по правилам записи имен для классов.

Рис. 12. Графическое изображение шаблона на диаграмме классов

Шаблон не может быть непосредственно использован в качестве класса, поскольку содержит неопределенные параметры. Чаще всего в качестве шаблона выступает некоторый суперкласс, параметры которого уточняются в его классах-потомках. Очевидно, в этом случае между ними существует отношение зависимости с ключевым словом "bind", когда класс-клиент может использовать некоторый шаблон для своей последующей параметризации.

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

Такой стереотипный подход позволяет существенно сократить сроки реализации проекта, однако приемлем лишь в том случае, когда новый проект концептуально и технологически не слишком отличается от предыдущих.

Моделирование Диаграммы состояний

Рассмотренная выше диаграмма классов представляет собой логическую модель статического представления моделируемой системы. Речь идет о том, что на данной диаграмме изображаются только взаимосвязи структурного характера, не зависящие от времени или реакции системы на внешние события. Однако для большинства физических систем, кроме самых простых и тривиальных, статических представлений совершенно недостаточно для моделирования процессов функционирования подобных систем как в целом, так и их отдельных подсистем и элементов.

Характеристика состояний системы не зависит (или слабо зависит) от логической структуры, зафиксированной в диаграмме классов. Поэтому при рассмотрении состояний системы приходится на время отвлечься от особенностей ее объектной структуры и мыслить категориями, образующими динамический контекст поведения моделируемой системы.

Каждая прикладная система характеризуется не только структурой составляющих ее элементов, но и некоторым поведением или функциональностью. Для общего представления функциональности моделируемой системы предназначены диаграммы вариантов использования, которые на концептуальном уровне описывают поведение системы в целом.

Для моделирования поведения на логическом уровне в языке UML могут использоваться сразу несколько канонических диаграмм: состояний, деятельности, последовательности и кооперации, каждая из которых 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. При этом изменение состояния объекта может быть вызвано внешними воздействиями со стороны других объектов или извне. Именно для описания реакции объекта на подобные внешние воздействия и используются диаграммы состояний.

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

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

Диаграмма состояний по существу является графом специального вида, который представляет некоторый автомат. Понятие автомата в контексте UML обладает довольно специфической семантикой, основанной на теории автоматов. Вершинами этого графа являются состояния и некоторые другие типы элементов автомата (псевдосостояния), которые изображаются соответствующими графическими символами. Дуги графа служат для обозначения переходов из состояния в состояние. Диаграммы состояний могут быть вложены друг в друга, образуя вложенные диаграммы более детального представления отдельных элементов модели. Для понимания семантики конкретной диаграммы состояний необходимо представлять не только особенности поведения моделируемой сущности, но и знать общие сведения по теории автоматов.

Автоматы

Автомат (state machine) в языке UML представляет собой некоторый формализм для моделирования поведения элементов модели и системы в целом. 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. С другой стороны, автомат описывает поведение отдельного объекта в форме последовательности состояний, которые охватывают все этапы его жизненного цикла, начиная от создания объекта и заканчивая его уничтожением. Каждая диаграмма состояний представляет некоторый автомат.

Простейшим примером визуального представления состояний и переходов на основе формализма автоматов может служить рассмотренная выше ситуация с исправностью технического устройства, такого как компьютер. В этом случае вводятся в рассмотрение два самых общих состояния: "исправен" и "неисправен" и два перехода: "выход из строя" и "ремонт". Графически эта информация может быть представлена в виде изображенной ниже диаграммы состояний компьютера (рис. 1).

Рис. 1. Простейший пример диаграммы состояний для технического устройства типа компьютер

Основными понятиями, входящими в формализм автомата, являются состояние и переход. Главное различие между ними заключается в том, что длительность нахождения системы в отдельном состоянии существенно превышает время, которое затрачивается на переход из одного состояния в другое. Предполагается, что в пределе время перехода из одного состояния в другое равно нулю (если дополнительно ничего не сказано). Другими словами, переход объекта из состояния в состояние происходит мгновенно.

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

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

Еще одним свойством графа состояний может служить достижимость состояний. Речь идет о том, что навигация или ориентированный путь в графе состояний определяет специальное бинарное отношение на множестве всех состояний системы. Это отношение характеризует потенциальную возможность перехода системы из рассматриваемого состояния в некоторое другое состояние. Очевидно, для достижимости состояний необходимо наличие связывающего их ориентированного пути в графе состояний.

Формализм автоматов допускает вложение одних автоматов в другие для уточнения внутренней структуры отдельных более общих состояний (макросостояний). В этом случае вложенные автоматы получили название подавтоматов. Подавтоматы могут использоваться для внутренней спецификации процедур и функций, образующих поведение исходного объекта.

Формализм обычного автомата основан на выполнении следующих обязательных условий:

1. Автомат не запоминает историю перемещения из состояния в состояние. С точки зрения моделируемого поведения определяющим является сам факт нахождения объекта в том или ином состоянии, но никак не последовательность состояний, в результате которой объект перешел в текущее состояние.
2. В каждый момент времени автомат может находиться в одном и только в одном из своих состояний. Это означает, что формализм автомата предназначен для моделирования последовательного поведения, когда объект в течение своего жизненного цикла последовательно проходит через все свои состояния. При этом автомат может находиться в отдельном состоянии как угодно долго, если не происходит никаких событий.
3. Хотя процесс изменения состояний автомата происходит во времени, явно концепция времени не входит в формализм автомата. Это означает, что длительность нахождения автомата в том или ином состоянии, а также время достижения того или иного состояния никак не специфицируются.
4. Количество состояний автомата должно быть обязательно конечным (в языке UML рассматриваются только конечные автоматы), и все они должны быть специфицированы явным образом. При этом 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.
5. Граф автомата не должен содержать изолированных состояний и переходов. Это условие означает, что для каждого из состояний, кроме начального, должно быть определено предшествующее состояние. Каждый переход должен обязательно соединять два состояния автомата. Допускается переход из состояния в себя, такой переход еще называют "петлей".
6. Автомат не должен содержать конфликтующих переходов, т. е. таких переходов из одного и того же состояния, когда объект одновременно может перейти в два и более последующих состояния (кроме случая параллельных подавтоматов). В языке UML исключение конфликтов возможно на основе введения так называемых сторожевых условий, которые будут рассмотрены ниже.

Состояние

Понятие состояния (state) является фундаментальным не только в метамоде-ли языка UML, но и в прикладном системном анализе. Вся концепция динамической системы основывается на понятии состояния системы.

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.

Состояние на диаграмме изображается прямоугольником со скругленными вершинами (рис. 2). Этот прямоугольник, в свою очередь, может быть разделен на две секции горизонтальной линией. Если указана лишь одна секция, то в ней записывается только имя состояния (рис. 2, а). В противном случае в первой из них записывается имя состояния, а во второй - список некоторых внутренних действий или переходов в данном состоянии (рис. 2, б). При этом под действием в языке UML понимают некоторую атомарную операцию, выполнение которой приводит к изменению состояния или возврату некоторого значения (например, "истина" или "ложь").

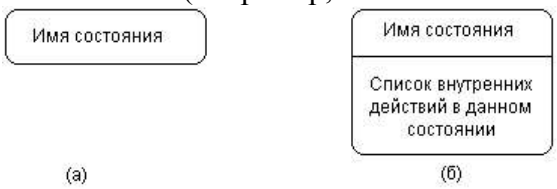

Рис.2. Графическое изображение состояний на диаграмме состояний

Имя состояния представляет собой строку текста, которая раскрывает содержательный смысл данного состояния. Имя всегда записывается с заглавной буквы. Поскольку состояние системы является составной частью процесса ее функционирования, рекомендуется в качестве имени использовать глаголы в настоящем времени (звенит, печатает, ожидает) или соответствующие причастия (занят, свободен, передано, получено). Имя у состояния может отсутствовать, т. е. оно является необязательным для некоторых состояний.

Список внутренних действий. Эта секция содержит перечень внутренних действий или деятельностей, которые выполняются в процессе 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. При этом выражение действия может использовать любые атрибуты и связи, которые принадлежат области имен или контексту моделируемого объекта. Если список выражений действия пустой, то разделитель в виде наклонной черты '/' может не указываться.

В качестве примера состояния рассмотрим ситуацию ввода пароля пользователя при аутентификации входа в некоторую программную систему (рис. 3). В этом случае список внутренних действий в данном состоянии не пуст и включает 4 отдельных действия, первые два из которых стандартные и описаны выше, а два последних определяются своей спецификацией.

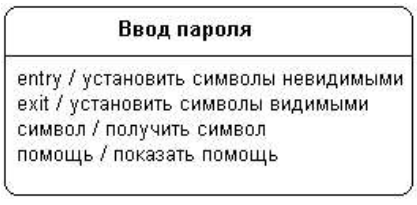

Рис. 3. Пример состояния с непустой секцией внутренних действий

Начальное состояние. Начальное состояние представляет собой частный случай состояния, которое не содержит никаких внутренних действий (псевдосостояния). В этом состоянии находится объект по умолчанию в начальный момент времени. Оно служит для указания на диаграмме состояний графической области, от которой начинается процесс изменения состояний. Графически начальное состояние в языке UML обозначается в виде закрашенного кружка (рис. 4, а), из которого может только выходить стрелка, соответствующая переходу.

Рис. 4. Графическое изображение начального и конечного состояний на диаграмме состояний

На самом верхнем уровне представления объекта переход из начального состояния может быть помечен событием создания (инициализации) данного объекта. В противном случае переход никак не помечается. Если этот переход не помечен, то он является первым переходом в следующее за ним состояние.

Конечное состояние. Конечное (финальное) состояние представляет собой частный случай состояния, которое также не содержит никаких внутренних действий (псевдосостояния). В этом состоянии будет находиться объект по умолчанию после завершения работы автомата в конечный момент времени. Оно служит для указания на диаграмме состояний графической области, в которой завершается процесс изменения состояний или жизненный цикл данного объекта. Графически конечное состояние в языке UML обозначается в виде закрашенного кружка, помещенного в окружность (рис. 4, б), в которую может только входить стрелка, соответствующая переходу.

Переход

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 (не путать с начальным состоянием - это разные понятия), 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 Кроме того, на переходе могут указываться 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".

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

На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние (например, "выход из строя" на рис. 1). Каждый переход может 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>')'.

Термин **событие (event)** требует отдельного пояснения, поскольку является самостоятельным элементом языка UML. Формально, событие представляет собой спецификацию некоторого факта, имеющего место в пространстве и во времени. Про события говорят, что они "происходят", при этом отдельные события должны быть упорядочены во времени. После наступления некоторого события нельзя уже вернуться к предыдущим событиям, если такая возможность не предусмотрена явно в модели.

Семантика понятия события фиксирует внимание на внешних проявлениях качественных изменений, происходящих при переходе моделируемого объекта из состояния в состояние. Например, при включении электрического переключателя происходит некоторое событие, в результате которого комната становится освещенной.

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. Например, переходы на рис. 1 являются триггерными, поскольку с каждым из них связано некоторое событие-триггер, происходящее асинхронно в момент выхода из строя технического устройства или в момент окончания его ремонта.

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. Напомним, что булевское выражение должно принимать одно их двух взаимно исключающих значений: "истина" или "ложь". Из контекста диаграммы состояний должна явно следовать семантика этого выражения, а для записи выражения может использоваться синтаксис языка объектных ограничений, основы которого изложены в приложении.

Введение для перехода сторожевого условия позволяет явно специфицировать семантику его срабатывания. 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 в том и только в том случае, когда переход срабатывает. Представляет собой атомарную операцию (достаточно простое вычисление), 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 оказывать влияние как на сам объект, так и на его окружение, если это с очевидностью следует из контекста модели. Выражение записывается после знака "/" в строке текста, присоединенной к соответствующему переходу.

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

Моделирование Диаграммы 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.

Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Важно подчеркнуть то обстоятельство, что с увеличением сложности системы строгое соблюдение последовательности выполняемых операций приобретает все более важное значение.

Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. 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.

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. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения.

Хотя диаграмма деятельности предназначена для моделирования поведения систем, время в явном виде отсутствует на этой диаграмме.

Состояние действия

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

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

Рис. 1. Графическое изображение состояния действия

Действие может быть записано на естественном языке, некотором псевдокоде или языке программирования. Никаких дополнительных или неявных ограничений при записи действий не накладывается. Рекомендуется в качестве имени простого действия использовать глагол с пояснительными словами (рис. 1, а). Если же действие может быть представлено в некотором формальном виде, то целесообразно записать его на том языке программирования, на котором предполагается реализовывать конкретный проект (рис. 1, б).

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

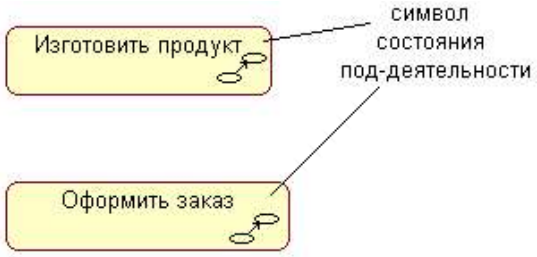

Рис. 2. Графическое изображение состояния под-деятельности

Каждая диаграмма деятельности должна иметь единственное начальное и единственное конечное состояния. При этом каждая деятельность начинается в начальном состоянии и заканчивается в конечном состоянии. Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали сверху вниз. В этом случае начальное состояние будет изображаться в верхней части диаграммы, а конечное - в ее нижней части.

Переходы

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

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

Графически ветвление на диаграмме деятельности обозначается небольшим ромбом, внутри которого нет никакого текста. В этот ромб может входить только одна стрелка от

того состояния действия, после выполнения которого поток управления должен быть продолжен по одной из взаимно исключающих ветвей. Принято входящую стрелку присоединять к верхней или левой вершине символа ветвления. Выходящих стрелок может быть две или более, но для каждой из них явно указывается соответствующее сторожевое условие в форме булевского выражения.

В примере (рис. 3) рассчитывается общая стоимость товаров, покупаемых по кредитной карточке в супермаркете. Если эта стоимость превышает $50, то выполняется аутентификация личности владельца карточки. В случае положительной проверки (карточка действительная) или если стоимость товаров не превышает $50, происходит снятие суммы со счета и оплата стоимости товаров. При отрицательном результате (карточка недействительная) оплаты не происходит, и товар остается у продавца.

Как видно из этого же рисунка, допускается использовать вместо сторожевого условия слово "иначе", указывающее на тот переход, который должен сработать в случае невыполнения сторожевого условия ветвления.

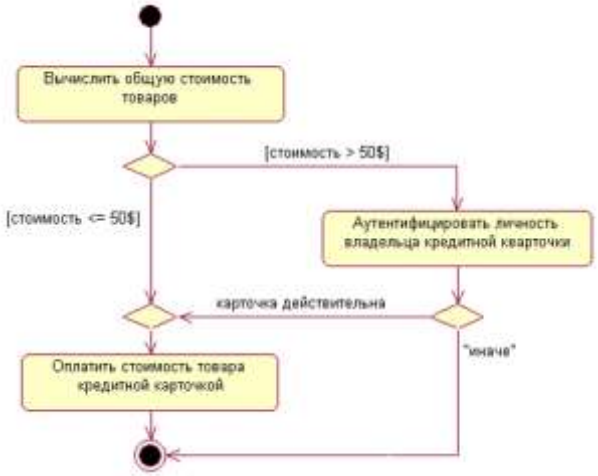

Рис. 3 Различные варианты ветвлений на диаграмме деятельности

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.

Как правило, такая черточка изображается отрезком горизонтальной линии, толщина которой несколько шире основных сплошных линий диаграммы деятельности. При этом разделение (concurrent fork) имеет один входящий переход и несколько выходящих (рис. 4, а). Слияние (concurrent join), наоборот, имеет несколько входящих переходов и один выходящий (рис. 4, б).

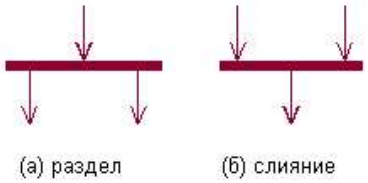

Рис. 4. Графическое изображение разделения и слияния параллельных потоков управления

Дорожки

Диаграммы деятельности могут быть использованы не только для спецификации алгоритмов вычислений или потоков управления в программных системах. Не менее важная

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

Для моделирования этих особенностей в языке UML используется специальная конструкция, получившее название дорожки (swimlanes). Имеется в виду визуальная аналогия с плавательными дорожками в бассейне, если смотреть на соответствующую диаграмму. При этом все состояния действия на диаграмме деятельности делятся на отдельные группы, которые отделяются друг от друга вертикальными линиями. Две соседние линии и образуют дорожку, а группа состояний между этими линиями выполняется отдельным подразделением (отделом, группой, отделением, филиалом) компании (рис. 5).

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.

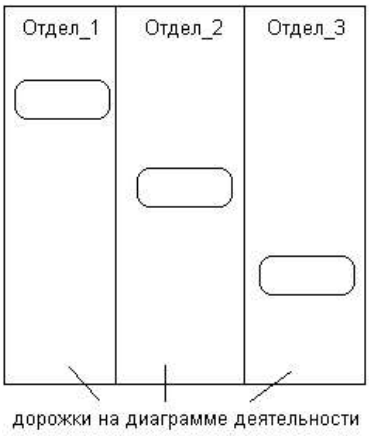

Рис. 5. Вариант диаграммы деятельности с дорожками

В качестве примера рассмотрим фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов по телефону. Подразделениями компании являются отдел приема и оформления заказов, отдел продаж и склад.

Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения. В данном случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое из подразделений торговой компании должно выполнять то или иное действие (рис. 6).

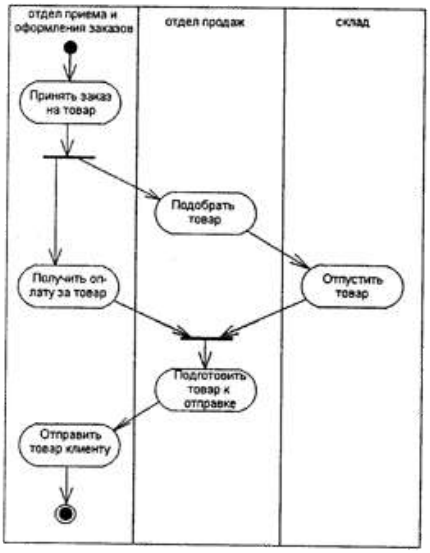

Рис. 6. Фрагмент диаграммы деятельности для торговой компании

Объекты

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

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

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

Возвращаясь к предыдущему примеру с торговой компанией, можно заметить, что центральным объектом процесса продажи является заказ или вернее состояние его выполнения. Вначале до звонка от клиента заказ как объект отсутствует и возникает лишь после такого звонка. Однако этот заказ еще не заполнен до конца, поскольку требуется еще подобрать конкретный товар в отделе продаж. После его подготовки он передается на склад, где вместе с отпуском товара заказ окончательно дооформляется. Наконец, после получения подтверждения об оплате товара эта информация заносится в заказ, и он считается выполненным и закрытым. Данная информация может быть представлена графически в виде модифицированного варианта диаграммы деятельности этой же торговой компании (рис. 7).

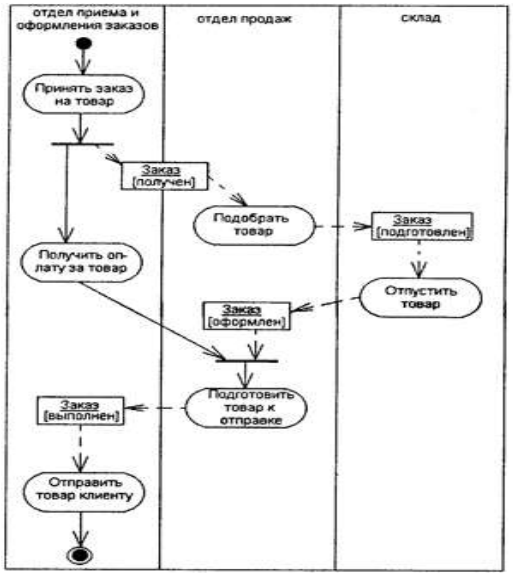

Рис. 7. Фрагмент диаграммы деятельности торговой компании с объектом-заказом

В заключение следует остановиться на необходимости синхронизации отдельных действий на диаграмме деятельности. Такая необходимость возникает всякий раз, когда параллельно выполняемые действия оказывают влияние на друг на друга. На диаграмме деятельности никаких дополнительных обозначений не используется, поскольку синхронизация параллельных процессов может быть реализована с помощью переходов "разделение-слияние".

Диаграммы деятельности играют важную роль в понимании процессов реализации алгоритмов выполнения операций классов и потоков управления в моделируемой системе. Используемые для этой цели традиционные блок-схемы алгоритмов обладают серьёзными ограничениями в представлении параллельных процессов и их синхронизации. Применение дорожек и объектов открывает дополнительные возможности для наглядного представления бизнес-процессов, позволяя специфицировать деятельность подразделений компаний и фирм.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. На основе приведённых примеров рассмотреть основные принципы построения диаграмм классов, состояния и деятельности на основе языка UML.
3. По варианту задания построить диаграмму классов.
4. Построить диаграмму состояний.
5. Построить диаграмму деятельнсти.
6. В электронном виде оформите краткий отчет о выполненной работе.
7. Сдать выполненную работу.

Форма представления результата:

Построенные диаграммы, отчет о работе.

Критерии оценки:

Отлично - работа выполнена полностью, без ошибок, сделан и сдан отчет.

Хорошо - работа выполнена полностью, с незначительными ошибками или неточностями, сделан и сдан отчет.

Удовлетворительно - работа выполнена не полностью, со незначительными ошибками или неточностями, отчет не сделан или сделан частично.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 4 Построение диаграммы компонентов и генерация кода

Цель: ознакомиться с методологией моделирования диаграммы компонентов и на основе языка UML

Выполнив работу, Вы будете:

уметь:

- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств.
- проектировать и разрабатывать систему по заданным требованиям и спецификациям.
- использовать методологию языка UML для моделирования диаграммы компонентов.

Материальное обеспечение:

Персональный компьютер

Задание:

1. Ознакомиться с теоретическим материалом.
2. Ознакомиться с методологией построения диаграммы компонентов
3. Выполнить построение диаграммы компонентов.
4. Сдайте выполненную работу.

Краткие теоретические сведения:

Все рассмотренные ранее диаграммы отражали концептуальные аспекты построения модели системы и относились к логическому уровню представления. Особенность логического представления заключается в том, что оно оперирует понятиями, которые не имеют самостоятельного материального воплощения. Другими словами, различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

Основное назначение логического представления состоит в анализе структурных и функциональных отношений между элементами модели системы. Однако для создания конкретной физической системы необходимо некоторым образом реализовать все элементы логического представления в конкретные материальные сущности. Для описания таких реальных сущностей предназначен другой аспект модельного представления, а именно физическое представление модели.

Чтобы пояснить отличие логического и физического представлений, рассмотрим в общих чертах процесс разработки некоторой программной системы. Ее исходным **логическим** представлением могут служить структурные схемы алгоритмов и процедур, описания интерфейсов и концептуальные схемы баз данных.

Необходимыми элементами **физического** представления системы являются исполняемые модули, библиотеки классов и процедур, стандартных графических интерфейсов, файлов баз данных.

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

Диаграмма компонентов, в отличие от ранее рассмотренных диаграмм, описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между

программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

1. Визуализации общей структуры исходного кода программной системы.
2. Спецификации исполнимого варианта программной системы.
3. Обеспечения многократного использования отдельных фрагментов программного кода.
4. Представления концептуальной и физической схем баз данных.

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

Компоненты

Для представления физических сущностей в языке UML применяется специальный термин - компонент (component). Компонент реализует некоторый набор интерфейсов и служит для общего обозначения элементов физического представления модели. Для графического представления компонента может использоваться специальный символ - прямоугольник со вставленными слева двумя более мелкими прямоугольниками (рис. 1). Внутри объемлющего прямоугольника записывается имя компонента и, возможно, некоторая дополнительная информация. Изображение этого символа может незначительно варьироваться в зависимости от характера ассоциируемой с компонентом информации.

В метамодели языка UML компонент является потомком классификатора. Он предоставляет организацию в рамках физического пакета ассоциированным с ним элементам модели. Как классификатор, компонент может иметь также свои собственные свойства, такие как атрибуты и операции.

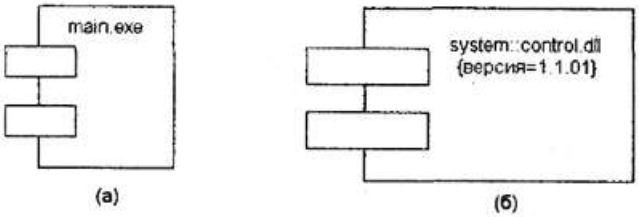

Рис. 1. Графическое изображение компонента в языке UML

Так, в первом случае (рис. 1, а) с компонентом уровня экземпляра связывается только его имя, а во втором (рис. 1, б) - дополнительно имя пакета и помеченное значение.

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и некоторых знаков препинания. Отдельный компонент может быть представлен на уровне типа или на уровне экземпляра. Хотя его графическое изображение в обоих случаях одинаковое, правила записи имени компонента несколько отличаются. Если компонент представляется на уровне типа, то в качестве его имени записывается только имя типа с заглавной буквы.

Если же компонент представляется на уровне экземпляра, то в качестве его имени записывается <имя компонента ':' имя типаX При этом вся строка имени подчеркивается.

В качестве простых имен принято использовать имена исполняемых файлов (с указанием расширения exe после точки-разделителя), имена динамических библиотек

(расширение dll), имена Web-страниц (расширение html), имена текстовых файлов (расширения txt или doc) или файлов справки (hip), имена файлов баз данных (DB) или имена файлов с исходными текстами программ (расширения h, cpp для языка C++, расширение Java для языка Java), скрипты (pi, asp) и др.

Поскольку конкретная реализация логического представления модели системы зависит от используемого программного инструментария, то и имена компонентов будут определяться особенностями синтаксиса соответствующего языка программирования.

Виды компонентов

Поскольку компонент как элемент физической реализации модели представляет отдельный модуль кода, иногда его комментируют с указанием дополнительных графических символов, иллюстрирующих конкретные особенности его реализации. Строго говоря, эти дополнительные обозначения для примечаний не специфицированы в языке UML. Однако их применение упрощает понимание диаграммы компонентов, существенно повышая наглядность физического представления. Некоторые из таких общепринятых обозначений для компонентов изображены ниже (рис. .2).

В языке UML выделяют три вида компонентов.

- Во-первых, компоненты развертывания, которые обеспечивают непосредственное выполнение системой своих функций. Такими компонентами могут быть динамически подключаемые библиотеки с расширением dll (рис. 2, а), Web-страницы на языке разметки гипертекста с расширением html (рис. .2, б) и файлы справки с расширением Ыр (рис. 2, в).
- Во-вторых, компоненты-рабочие продукты. Как правило - это файлы с исходными текстами программ, например, с расширениями h или cpp для языка C++ (рис. 2, г).
- В-третьих, компоненты исполнения, представляющие исполнимые модули - файлы с расширением exe. Они обозначаются обычным образом.

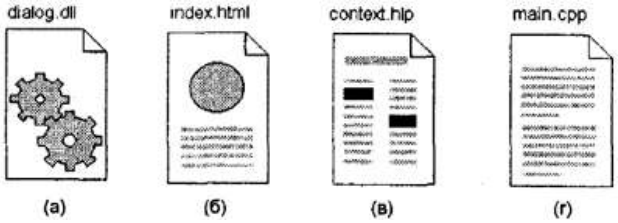

Рис. 2. Варианты графического изображения компонентов на диаграмме компонентов

Интерфейсы

Следующим элементом диаграммы компонентов являются интерфейсы. В общем случае интерфейс графически изображается окружностью, которая соединяется с компонентом отрезком линии без стрелок (рис. 3, а). При этом имя интерфейса, которое обязательно должно начинаться с заглавной буквы "I", записывается рядом с окружностью. Семантически линия означает реализацию интерфейса, а наличие интерфейсов у компонента означает, что данный компонент реализует соответствующий набор интерфейсов.

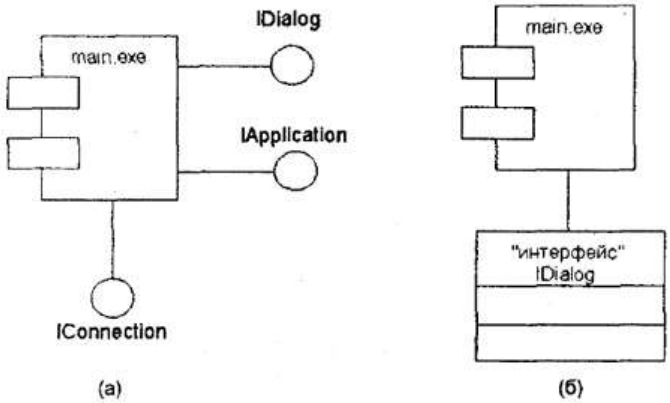

Рис. 3. Графическое изображение интерфейсов на диаграмме компонентов

Другим способом представления интерфейса на диаграмме компонентов является его изображение в виде прямоугольника класса со стереотипом "интерфейс" и возможными секциями атрибутов и операций (рис. 3, б). Как правило, этот вариант обозначения используется для представления внутренней структуры интерфейса, которая может быть важна для реализации.

При разработке программных систем интерфейсы обеспечивают не только совместимость различных версий, но и возможность вносить существенные изменения в одни части программы, не изменяя другие ее части. Таким образом, назначение интерфейсов существенно шире, чем спецификация взаимодействия с пользователями системы (актерами).

Зависимости

Зависимость не является ассоциацией, а служит для представления только факта наличия такой связи, когда изменение одного элемента модели оказывает влияние или приводит к изменению другого элемента модели. Отношение зависимости на диаграмме компонентов изображается пунктирной линией со стрелкой, направленной от клиента (зависимого элемента) к источнику (независимому элементу).

Зависимости могут отражать связи модулей программы на этапе компиляции и генерации объектного кода. В другом случае зависимость может отражать наличие в независимом компоненте описаний классов, которые используются в зависимом компоненте для создания соответствующих объектов. Применительно к диаграмме компонентов зависимости могут связывать компоненты и импортируемые этим компонентом интерфейсы, а также различные виды компонентов между собой.

В первом случае рисуют стрелку от компонента-клиента к импортируемому интерфейсу (рис. 4). Наличие такой стрелки означает, что компонент не реализует соответствующий интерфейс, а использует его в процессе своего выполнения. Причем на этой же диаграмме может присутствовать и другой компонент, который реализует этот интерфейс. Так, например, изображенный ниже фрагмент диаграммы компонентов представляет информацию о том, что компонент с именем "main.exe" зависит от импортируемого интерфейса I Dialog, который, в свою очередь, реализуется компонентом с именем "image.java". Для второго компонента этот же интерфейс является экспортируемым.

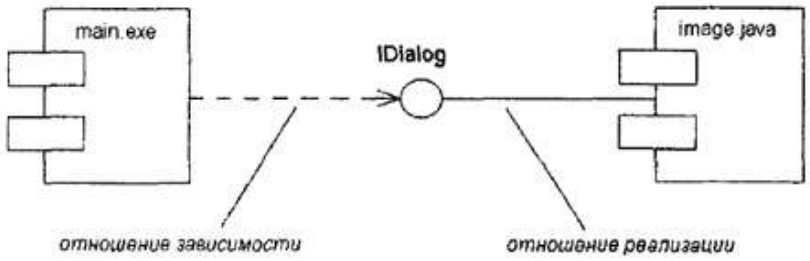

Рис. 4. Фрагмент диаграммы компонентов с отношением зависимости

Другим случаем отношения зависимости на диаграмме компонентов является отношение между различными видами компонентов (рис. 5). Наличие подобной зависимости означает, что внесение изменений в исходные тексты программ или динамические библиотеки приводит к изменениям самого компонента. При этом характер изменений может быть отмечен дополнительно.

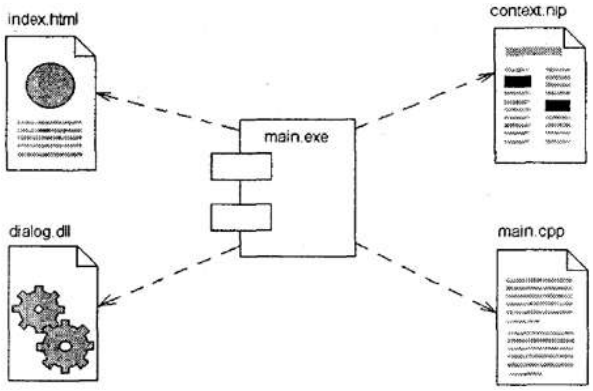

Рис. 5. Графическое изображение отношения зависимости между компонентами

Наконец, на диаграмме компонентов могут быть представлены отношения зависимости между компонентами и реализованными в них классами. Эта информация имеет важное значение для обеспечения согласования логического и физического представлений модели системы. Разумеется, изменения в структуре описаний классов могут привести к изменению компонента. Ниже приводится фрагмент зависимости подобного рода, когда некоторый компонент зависит от соответствующих классов.

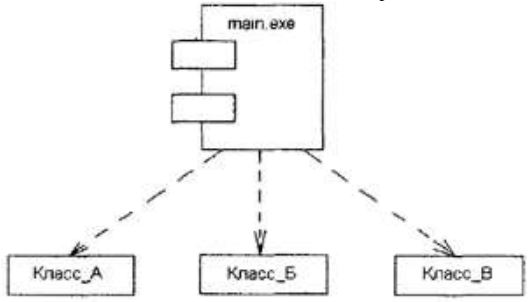

Рис. 6. Графическое изображение зависимости между компонентом и классами

Следует заметить, что в данном случае из диаграммы компонентов не следует, что классы реализованы этим компонентом. Если требуется подчеркнуть, что некоторый компонент реализует отдельные классы, то для обозначения компонента используется расширенный символ прямоугольника. При этом прямоугольник компонента делится на две секции горизонтальной линией. Верхняя секция служит для записи имени компонента, а нижняя секция - для указания дополнительной информации (рис. 7).

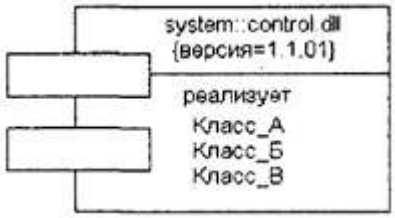

Рис. 7. Графическое изображение компонента с дополнительной информацией о реализуемых им классах

Внутри символа компонента могут изображаться другие элементы графической нотации, такие как классы (компонент уровня типа) или объекты (компонент уровня экземпляра). В этом случае символ компонента изображается таким образом, чтобы вместить эти дополнительные символы. Так, например, изображенный ниже компонент (рис. 8) является экземпляром и реализует три отдельных объекта.

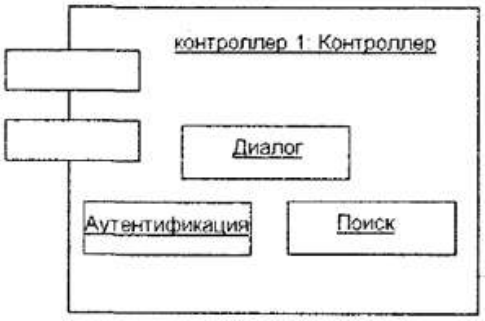

Рис. 8. Графическое изображение компонента уровня экземпляра, реализующего отдельные объекты

Объекты, которые находятся в отдельном компоненте-экземпляре, изображаются вложенными в символ данного компонента. Подобная вложенность означает, что выполнение компонента влечет выполнение соответствующих объектов. Другими словами, существование компонента в течение времени исполнения программы обеспечивает существование, а возможно, и доступ всех вложенных в него объектов. Что касается доступа к этим объектам, то он может быть дополнительно специфицирован с помощью квантификаторов видимости, подобно видимости пакетов. Содержательный смысл видимости может отличаться для различных видов пакетов.

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

После этого можно приступать к общей структуризации диаграммы компонентов. В первую очередь, необходимо решить, из каких физических частей (файлов) будет состоять программная система. На этом этапе следует обратить внимание на такую реализацию системы, которая обеспечивала бы не только возможность повторного использования кода за счет рациональной декомпозиции компонентов, но и создание объектов только при их необходимости.

После общей структуризации физического представления системы необходимо дополнить модель интерфейсами и схемами базы данных. При разработке интерфейсов следует обращать внимание на согласование (стыковку) различных частей программной системы. Включение в модель схемы базы данных предполагает спецификацию отдельных таблиц и установление информационных связей между таблицами.

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

Диаграмма компонентов, как правило, разрабатывается совместно с диаграммой развертывания, на которой представляется информация о физическом размещении компонентов программной системы по ее отдельным узлам. Особенности построения диаграммы развертывания будут рассмотрены в следующей главе.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. На основе приведённых примеров рассмотреть основные принципы построения диаграммы компонентов.

3. По варианту задания построить диаграмму компонентов для соответствующей информационной системы.
4. В электронном виде оформите краткий отчет о выполненной работе.
5. Сдать выполненную работу.

Форма представления результата:

Построенные диаграммы, отчет о работе.

Критерии оценки:

Отлично - работа выполнена полностью, без ошибок, сделан и сдан отчет.

Хорошо - работа выполнена полностью, с незначительными ошибками или неточностями, сделан и сдан отчет.

Удовлетворительно - работа выполнена не полностью, со незначительными ошибками или неточностями, отчет не сделан или сделан частично.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 5 Построение диаграммы потоков данных и генерация кода

Цель: ознакомиться с методологией моделирования диаграммы потоков данных и на основе языка UML

Выполнив работу, Вы будете:

уметь:

- осуществлять постановку задач по обработке информации;
- проводить анализ предметной области;
- осуществлять выбор модели и средства построения информационной системы и программных средств.
- проектировать и разрабатывать систему по заданным требованиям и спецификациям.
- использовать методологию языка UML для моделирования диаграммы потоков.

Материальное обеспечение:

Персональный компьютер

Задание:

1. Ознакомиться с теоретическим материалом.
2. Ознакомиться с методологией построения диаграммы потоков данных
3. Выполнить построение диаграммы потоков данных.
4. Сдайте выполненную работу.

Краткие теоретические сведения:

Модель системы в диаграмме потоков данных представляется в виде некоторой информационной модели, основными компонентами которой являются различные потоки данных, которые переносят информацию от одной подсистемы к другой. Каждая из подсистем выполняет определенные преобразования входного потока данных и передает результаты обработки информации в виде потоков данных для других подсистем.

Основными компонентами диаграмм потоков данных являются:

1. внешние сущности
2. накопители данных или хранилища
3. процессы
4. потоки данных
5. системы/подсистемы

Внешняя сущность представляет собой материальный объект или физическое лицо, которые могут выступать в качестве источника или приемника информации. Определение некоторого объекта или системы в качестве внешней сущности не является строго фиксированным. Хотя внешняя сущность находится за пределами границ рассматриваемой системы, в процессе дальнейшего анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы модели системы. С другой стороны, отдельные процессы могут быть вынесены за пределы диаграммы и представлены как внешние сущности. Примерами внешних сущностей могут служить: клиенты организации, заказчики, персонал, поставщики.

Внешняя сущность обозначается прямоугольником с тенью (рис. 2.15), внутри которого указывается ее имя. При этом в качестве имени рекомендуется использовать существительное в именительном падеже. Иногда внешнюю сущность называют также терминатором.

Рис. 1 Изображение внешней сущности на диаграмме потоков данных

Накопитель данных или хранилище представляет собой абстрактное устройство или способ хранения информации, перемещаемой между процессами. Предполагается, что данные можно в любой момент поместить в накопитель и через некоторое время извлечь, причем физические способы помещения и извлечения данных могут быть произвольными. Накопитель данных может быть физически реализован различными способами, но наиболее часто предполагается его реализация в электронном виде на магнитных носителях. Накопитель данных на диаграмме потоков данных изображается прямоугольником с двумя полями (рис. 2). Первое поле служит для указания номера или идентификатора накопителя, который начинается с буквы "D". Второе поле служит для указания имени. При этом в качестве имени накопителя рекомендуется использовать существительное, которое характеризует способ хранения соответствующей информации.

Рис. 2 Изображение накопителя на диаграмме потоков данных

Процесс представляет собой совокупность операций по преобразованию входных потоков данных в выходные в соответствии с определенным алгоритмом или правилом. Хотя физически процесс может быть реализован различными способами, наиболее часто подразумевается программная реализация процесса. Процесс на диаграмме потоков данных изображается прямоугольником с закругленными вершинами (рис. 3), разделенным на три секции или поля горизонтальными линиями. Поле номера процесса служит для идентификации последнего. В среднем поле указывается имя процесса. В качестве имени рекомендовано использовать глагол в неопределенной форме с необходимыми дополнениями. Нижнее поле содержит указание на способ физической реализации процесса.

Рис. 3 Изображение процесса на диаграмме потоков данных

Информационная модель системы строится как некоторая иерархическая схема в виде так называемой контекстной диаграммы, на которой исходная модель последовательно представляется в виде модели подсистем соответствующих процессов преобразования данных. При этом подсистема или система на контекстной диаграмме DFD изображается так же, как и процесс – прямоугольником с закругленными вершинами (рис. 4).

Рис. 4. Изображение подсистемы на диаграмме потоков данных

Поток данных определяет качественный характер информации, передаваемой через некоторое соединение от источника к приемнику. Реальный поток данных может передаваться по сети между двумя компьютерами или любым другим способом, допускающим извлечение данных и их восстановление в требуемом формате. Поток данных на диаграмме изображается линией со стрелкой на одном из ее концов, при этом стрелка

показывает направление потока данных. Каждый поток данных имеет свое собственное имя, отражающее его содержание.

Таким образом, информационная модель системы строится в виде диаграмм потоков данных, которые графически представляются с использованием соответствующей системы обозначений. В качестве примера рассмотрим упрощенную модель процесса получения некоторой суммы наличными по кредитной карточке клиентом банка. Внешними сущностями данного примера являются клиент банка и, возможно, служащий банка, который контролирует процесс обслуживания клиентов. Накопителем данных может быть база данных о состоянии счетов отдельных клиентов банка. Отдельные потоки данных отражают характер передаваемой информации, необходимой для обслуживания клиента банка. Соответствующая модель для данного примера может быть представлена в виде диаграммы потоков данных (рис. 5).

Рис. 5. Пример диаграммы потоков данныхдля процесса получения некоторой суммы наличными по кредитной карточке

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. На основе приведённых примеров рассмотреть основные принципы построения диаграммы потоков.
3. По варианту задания построить диаграмму потоков для соответствующей информационной системы.
4. В электронном виде оформите краткий отчет о выполненной работе.
5. Сдать выполненную работу.

Форма представления результата:

Построенные диаграммы, отчет о работе.

Критерии оценки:

Отлично - работа выполнена полностью, без ошибок, сделан и сдан отчет.

Хорошо - работа выполнена полностью, с незначительными ошибками или неточностями, сделан и сдан отчет.

Удовлетворительно - работа выполнена не полностью, со незначительными ошибками или неточностями, отчет не сделан или сделан частично.

Неудовлетворительно - работа не выполнена.

Тема 5.2.2. Разработка и модификация информационных систем

Практическая работа № 1 Обоснование выбора технических средств.

Цель: ознакомиться с основными принципами выбора технических средств в процессе проектирования информационной системы; научиться осуществлять обоснование и выбор технических средств

Выполнив работу, Вы будете:

уметь:

- проводить анализ предметной области;
- распознавать задачу и/или проблему в профессиональном и/или социальном контексте;
- осуществлять выбор модели и средства построения информационной системы;
- осуществлять выбор технических средств для работы программного обеспечения
- взаимодействовать с коллегами, руководством, клиентами в ходе профессиональной деятельности;
- понимать требования и оправдывать ожидания клиентов/работодателя

Материальное обеспечение:

Персональный компьютер

Задание:

1. Ознакомиться с теоретическим материалом.
2. Рассмотреть задание, составить описание предметной области
3. Создать документ обоснования выбора технических средств по конкретному заданию.
4. Сдайте выполненную работу.

Краткие теоретические сведения:

Общую структуру информационной системы можно рассматривать как совокупность обеспечивающих подсистем: технической и программной.

Техническое обеспечение - это комплекс технических средств, предназначенных для эксплуатации информационной системы. Основная сложность при выборе технического обеспечения - это то, что существует целый ряд прямых и обратных связей между системами технического и технологического обеспечения. Это проявляется как во влиянии комплекса аппаратно-программных средств на проектируемый технологический процесс, так и в обратном влиянии технологического процесса на аппаратно-программное обеспечение.

Также на выбор комплекса технических средств разработанной системы влияет набор функций, которые должна реализовывать система и методы хранения информации. Учесть все характеристики при выборе технических средств практически невозможно. Однако на основе анализа задач, алгоритмов их решения, входных потоков информации можно определить требования к набору основных технических характеристик, обеспечивающих эксплуатацию системы, и в дальнейшем составить перечень технических средств.

В первую очередь для этих целей проводится анализ предметной области. Определяются цели создания и назначение системы.

Среди факторов, влияющих на выбор технического обеспечения, можно выделить:

1. Совместимость - аппаратная и программная совместимость со стандартными платформами.

2. Состав технологических операций ввода/вывода данных (использование сканирующих устройств, спец. датчиков, графопостроителей, различных приводов).

3. Аппаратные требования используемого программного обеспечения.

4.Эргономические требования (например, разрешающая способность и частота обновления изображения монитора).

5. Экономические факторы (бюджетное ограничение на закупку техники).

6. Технологические требования к поддержке работы с ЛВС (наличие либо отсутствие сетевых контроллеров, стандарт и тип поддерживаемой сети).

7. Объемы обрабатываемых данных и требования к скорости их обработки.

Также среди факторов для выбора технического обеспечения системы можно перечислить: стоимость приобретения технических средств; стоимость использования технических средств; надежность технических средств и срок службы; удобство комплекса технических средств; производительность и быстродействие КТС и другие.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. Составить документ обоснования и выбора технических средств для эксплуатации информационной системы по конкретному заданию.
3. Структура и состав документа:
   1) Полное наименование системы.
   2) Цели и назначение системы.
      - В целях описать основные цели, которые должны быть достигнуты в процессе эксплуатации системы.
      - В назначении системы представить перечень решаемых задач.
   3) Структура комплекса технических средств.
      - Описать основные принципы работы системы, которые составляют техническую основу ее эксплуатации.
      - Составить перечень технических средств, обеспечивающих функционирование системы.
   4) Описание средств вычислительной техники с указанием назначения и базовых технических характеристик.
      Оценить характеристики и сделать выбор средств вычислительной техники, а также аппаратуры передачи данных. При оценке учитывать следующие факторы:
      - соответствие оборудования требованиям, предъявляемым системой к техническим и программным средствам;
      - стоимость оборудования, наличие технической поддержки, наличие на рынке специалистов с необходимым уровнем квалификации для обслуживания оборудования

      В электронном виде оформите краткий отчет о выполненной работе.
4. Сдать выполненную работу.

Форма представления результата:

Выполненная работа, отчет о работе.

Критерии оценки:

Отлично – анализ предметной области по заданию сделан правильно, документ обоснования и выбора технических средств выполнен по правилам, заполнен полностью, оформлен грамотно, учтены все разделы документа, работа сдана преподавателю.

Хорошо - анализ предметной области по заданию сделан правильно или с незначительными неточностями, документ обоснования и выбора технических средств выполнен по правилам, заполнен не полностью, оформлен грамотно, учтены все разделы документа, работа сдана преподавателю

Удовлетворительно - анализ предметной области по заданию сделан с ошибками, документ обоснования и выбора технических средств выполнен не по правилам, заполнен не полностью, оформлен безграмотно, учтены не все разделы документа, работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Практическая работа № 2 Стоимостная оценка проекта

Цель: ознакомиться с основными принципами стоимостной оценки проектов.

Выполнив работу, Вы будете:

уметь:

- проводить анализ предметной области;
- распознавать задачу и/или проблему в профессиональном и/или социальном контексте;
- применять знания по финансовой грамотности для профессиональной деятельности;
- выявлять достоинства и недостатки коммерческой идеи;
- взаимодействовать с коллегами, руководством, клиентами в ходе профессиональной деятельности;
- понимать требования и оправдывать ожидания клиентов/работодателя

Материальное обеспечение:

Персональный компьютер

Задание:

1. Ознакомиться с теоретическим материалом.
2. Рассмотреть задание.
3. Создать документ вычисления стоимостных оценок проекта.
4. Сдать выполненную работу.

Краткие теоретические сведения:

Оценка проекта может выполняться с различными целями:

- принятия решения о целесообразности проекта;
- сравнения вариантов систем в процессе выбора;
- проведения переговоров о стоимости проекта;
- планирования расходов на проект автоматизации (бюджетирование);
- контроля фактических расходов на проект автоматизации.

Оценка проекта может выполняться в нескольких направлениях: клиентами, поставщиками решений и инвесторами.

Проект считается успешным, если он завершен в установленные сроки, выполнен в рамках бюджета и в соответствии с ожиданиями заказчика.

Управление стоимостью проекта объединяет процессы, выполняемые в ходе планирования, разработки бюджета и контролирования затрат и обеспечивающие завершение проекта в рамках утвержденного бюджета. К процессам управления стоимостью относятся:

- стоимостная оценка – определение примерной стоимости ресурсов, необходимых для выполнения операций проекта;

- разработка бюджета расходов – суммирование оценок стоимости отдельных операций или пакетов работ с целью формирования базового плана по стоимости;

- управление стоимостью- воздействие на факторы, вызывающие отклонения по стоимости, и управление изменениями бюджета проекта.

Стоимостная оценка - это процесс установления стоимости ресурсов проекта, основанный на определенных фактах и допущениях. Для определения стоимостной оценки прежде всего необходимо определить пакет операций), длительность операций и требуемые ресурсы. Процесс оценки и его результат в значительной степени зависят от точности описания содержания, качества доступной информации, от стадии проекта.

На процесс стоимостной оценки оказывают влияние: время, отведенное для проведения оцениваемой операции, опыт менеджера, инструменты оценивания, заданная точность.

Оценка стоимости проекта начинается на предпроектной стадии (до заключения контракта) и выполняется в течение всего времени выполнения проекта.

Выделяют следующие оценки стоимости:

- оценка порядка величины; на предпроектной стадии первоначально может определяться только порядок величины стоимости (точность оценки порядка величины стоимости проекта от (-50%) до (+100%)).
- концептуальная оценка (точность концептуальной оценки находится в интервале (-30%) - (+50%)).
- предварительная оценка (точность предварительной оценки проекта колеблется от (-20%) до (+30%)).
- окончательная оценка (точность от (-15%) до (+20%)) .
- контрольная оценка (контрольная оценка имеет точность от -10% до +15%).

Таким образом, каждая последующая стадия жизненного цикла проекта имеет более точную стоимостную оценку

Стоимостная оценка обычно выражается в единицах валюты (доллары, рубли и т. д.) для облегчения сравнения проектов и операций внутри проекта.

Стоимость плановых операций оценивается для всех ресурсов, задействованных в проекте. К ресурсам относятся, в частности, специалисты, оборудование, телефонная связь, Интернет, арендованные помещения, а также особые статьи расходов, например учет уровня инфляции или расходы на непредвиденные обстоятельства.

Для процесса оценки стоимости необходимо иметь следующие входные данные:

-Факторы внешней среды предприятия. К факторам внешней среды относятся конъюнктура рынка, коммерческие базы данных и прайс-листы. Конъюнктура рынка - это рынок информационных систем, их конкурентная функциональность, стоимость, услуги на внедрение, сопровождение. Коммерческие базы данных и прайс-листы содержат сведения о квалификации и стоимости трудовых ресурсов, стоимости внедрения информационных систем.

- Активы организационного процесса - официальные и неофициальные правила, процедуры и руководства по стоимостной оценке, шаблоны стоимостной оценки, информация о стоимости ранее выполненных проектов.
- Описание содержания проекта содержит важную информацию о требованиях, ограничениях и допущениях проекта, которую необходимо учитывать при стоимостной оценке.
- Иерархическая структура работ определяет взаимоотношения между всеми элементами проекта и результатами проекта.
- Словарь ИСР содержит подробное описание работы для каждого элемента ИСР.
- План управления проектом - общий план мероприятий по исполнению, мониторингу и контролю над проектом, содержащий указания и руководства по составлению плана управления стоимостью и контролю за его исполнением, а также дополнительные планы.

Инструменты и методы, используемые для оценки стоимости

В зависимости от стадии проекта, необходимой степени точности, возможных расходов и трудозатрат применяются различные типы оценок стоимости:

1. **Оценка сверху-вниз** применяется на ранних стадиях в условиях недостаточной информации о проекте. Производится только одна оценка стоимости всего проекта на самом верхнем уровне. Такая оценка не требует много усилий, но имеет низкую точность.

2. **Оценка по аналогам** представляет вид оценки сверху-вниз. При этом используется фактическая стоимость ранее выполненных проектов для оценки текущего проекта. При наличии очень похожего проекта оценка может быть довольно точной. Такой тип оценки применяется на любом этапе жизненного цикла проекта. Оценка по аналогам не

требует много усилий при гарантированной точности, однако не всегда удается найти и определить схожие проекты. Точность оценки по аналогии колеблется от -30% до +50%. Стоимость подготовки такой оценки составляет 0,04%-0,15% от общей стоимости проекта.

3. **Оценка снизу-вверх** применяется на этапе подготовки базового плана проекта и формировании контрольной оценки. Процесс начинается с оценки деталей проекта с последующим суммированием деталей на итоговых уровнях. Степень точности оценки зависит от уровня детализации ИСР. Оценка снизу-вверх обеспечивает точность от +0,15/-10% до +5%/-5%, но имеет высокую стоимость (от 0,45% до 2% от общей стоимости проекта) и продолжительность.

4. **Параметрическая оценка** применяется на ранних тапах проекта. Процесс параметрической оценки состоит в определении параметров оцениваемого проекта, которые изменяются пропорционально стоимости проекта. На основании одного или нескольких параметров создается математическая модель. Например, в качестве параметра разработки программного обеспечения может быть выбрана стоимость разработки строки кода. Для оценки стоимости обследования может быть выбрано количество автоматизируемых бизнес-процессов. Наиболее распространенным параметром оценки стоимости IT-проектов является количество требуемого рабочего времени на выполнение операций (пакета операций). При тесной связи между стоимостью и параметрами проекта и при возможности точного измерения параметров можно увеличить точность расчетов. Преимущество данного метода: для оценки стоимости проекта достаточно знать "ставки" привлекаемых ресурсов: недостатком является низкая точность (-30%-+50%). Стоимость подготовки параметрической оценки составляет 0,04%-0,45% от общей стоимости проекта.

5. **Контрольные оценки** представляют собой разновидность оценок снизу-вверх. Контрольная оценка основана на принципе более детальной оценки снизу-вверх. При оценке затрат на работы проекта, как правило, определяют наиболее вероятное значение затрат, затраты при благоприятных и неблагоприятных обстоятельствах, то есть оптимистическую, пессимистическую и наиболее вероятную оценку. Для расчета математического ожидания и среднеквадратичного отклонения применяют формулы, которые используются в методике PERT:

Математическое ожидание = [оптимистическое + пессимистическое + (4х наиболее вероятное)]/6

Среднеквадратичное отклонение = (пессимистическое - оптимистическое)/ 6

Контрольные оценки обладают высокой точностью, применяются для формирования базового плана проекта, но их выполнение продолжительно и стоит довольно дорого.

6. **Анализ предложений исполнителей -** очень простой метод при условии наличия исполнителей и подрядных организаций, желающих выполнить данный объем работ. Техническое задание, тендерная или иная документация рассылается по исполнителям-претендентам с просьбой предоставить свои оценки стоимости (а зачастую — и продолжительности) выполнения данных работ.

Выходы процесса стоимостной оценки

Оценка стоимости операции - количественная оценка примерной стоимости ресурсов, необходимых для выполнения плановых операций. Она может предоставляться в сжатой форме или развернуто. Затраты оцениваются по всем ресурсам, использованным в оценке стоимости операции.

Вспомогательные данные, на основании которых была произведена стоимостная оценка, должны содержать описание содержания работ проекта для плановой операции:

- документацию того, как оценка получена;
- документацию обо всех допущениях, сделанных для оценки;
- документацию обо всех ограничениях.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. Составить документ стоимостной оценки своего варианта проекта по методам сверху – вниз и параметрической оценки.
3. Для стоимостной оценки методом «сверху-вниз» необходимо составить укрупненную оценку всего пакета работ по одному или нескольким показателям (по работам, исполнителям и др.). Данный метод оценки является предварительным, поэтому при оценке стоимости операций проекта часто используются нормативы. Нормироваться может стоимость единицы работ, поэтому для подсчета стоимости операции необходимо знать и объем работ на операции.
4. Для стоимостной оценки методом «параметрической оценки» необходимо выделить в проекте набор параметров, изменение которых повлияет на общую стоимость проекта и повлечёт пропорциональное её изменение. Математически параметрическая модель строится на основе одного или нескольких параметров. Оценку стоимости можно получить после того как построена модель и введены значения параметров. Если при оценке используют множество параметров, то каждому параметру необходимо присвоить свой весовой коэффициент, и оценка стоимости будет осуществляться согласно многопараметрической модели.
5. Сравните полученные по 2-м методам стоимостные оценки проекта, сделайте вывод.
6. В электронном виде оформите краткий отчет о выполненной работе.
7. Сдайте выполненную работу.

Форма представления результата:

Выполненные расчеты, отчет о работе.

Критерии оценки:

Отлично – выполнена стоимостная оценка проекта двумя методами, составлен отчет о работе, сделан анализ полученных результатов, работа сдана преподавателю.

Хорошо - – выполненная стоимостная оценка проекта двумя методами содержит неточности, составлен отчет о работе, сделан анализ полученных результатов, работа сдана преподавателю.

Удовлетворительно - выполненная стоимостная оценка проекта двумя методами содержит ошибки, отчет о работе составлен не полностью, сделан анализ полученных результатов, работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Практическая работа № 3 Построение и обоснование модели проекта

Цель: ознакомиться с основными принципами построения и обоснование модели проекта, научится выполнять построение модели проекта.

Выполнив работу, Вы будете:

уметь:

- проводить анализ предметной области;
- взаимодействовать с коллегами, руководством, клиентами в ходе профессиональной деятельности;
- анализировать задачу и/или проблему и выделять её составные части;
- определять задачи для поиска информации;
- определять необходимые источники информации;

Материальное обеспечение:

Персональный компьютер

Задание:

1. Ознакомиться с теоретическим материалом.
2. Выбрать модель жц системы для своего варианта.
3. Составить схему жц системы и описание процессов.
4. Создать отчет о выполненной работе.
5. Сдать выполненную работу.

Краткие теоретические сведения:

Жизненный цикл информационной системы — период времени, который начинается с момента принятия решения о необходимости создания информационной системы и заканчивается в момент ее полного изъятия из эксплуатации.

Понятие жизненного цикла является одним из базовых понятий методологии проектирования информационных систем. Методология проектирования информационных систем описывает процесс создания и сопровождения систем в виде жизненного цикла (ЖЦ) ИС, представляя его как некоторую последовательность стадий и выполняемых на них процессов.

Для каждого этапа определяются состав и последовательность выполняемых работ, получаемые результаты, методы и средства, необходимые для выполнения работ, роли и ответственность участников и т.д. Такое формальное описание ЖЦ ИС позволяет спланировать и организовать процесс коллективной разработки и обеспечить управление этим процессом.

Полный жизненный цикл информационной системы включает в себя, как правило:

- стратегическое планирование
- анализ
- проектирование
- реализацию
- внедрение
- эксплуатацию.

В общем случае жизненный цикл можно в свою очередь разбить на ряд стадий.

Стадии жизненного цикла ИС

Стадия — часть процесса создания ИС, ограниченная определенными временными рамками и заканчивающаяся выпуском конкретного продукта (моделей, программных компонентов, документации), определяемого заданными для данной стадии требованиями. Соотношение между процессами и стадиями также определяется используемой моделью жизненного цикла ИС.

Жизненный цикл информационной системы подразделяется на четыре стадии, границы каждой стадии определены некоторыми моментами времени, в которые необходимо принимать определенные критические решения и, следовательно, достигать определенных ключевых целей:

1) Начальная стадия. Устанавливается область применения системы и определяются граничные условия. Для этого необходимо идентифицировать все внешние объекты, с которыми должна взаимодействовать разрабатываемая система, и определить характер этого взаимодействия на высоком уровне. На начальной стадии идентифицируются все функциональные возможности системы и производится описание наиболее существенных из них.

2) Стадия уточнения. На стадии уточнения проводится анализ прикладной области, разрабатывается архитектурная основа информационной системы. При принятии любых решений, касающихся архитектуры системы, необходимо принимать во внимание разрабатываемую систему в целом. Это означает, что необходимо описать большинство функциональных возможностей системы и учесть взаимосвязи между отдельными ее составляющими. В конце стадии уточнения проводится анализ архитектурных решений и способов устранения главных факторов риска в проекте.

3) Стадия конструирования На стадии конструирования разрабатывается законченное изделие, готовое к передаче пользователю. По окончании этой стадии определяется работоспособность разработанного программного обеспечения.

4) Стадия передачи в эксплуатацию На стадии передачи в эксплуатацию разработанное программное обеспечение передается пользователям. При эксплуатации разработанной системы в реальных условиях часто возникают различного рода проблемы, которые требуют дополнительных работ по внесению корректив в разработанный продукт. Это, как правило, связано с обнаружением ошибок и недоработок. В конце стадии передачи в эксплуатацию необходимо определить, достигнуты цели разработки или нет.

Модели жизненного цикла

Модель жизненного цикла ИС — структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении жизненного цикла. Модель жизненного цикла зависит от специфики, масштаба и сложности проекта и специфики условий, в которых система создается и функционирует.

Модель ЖЦ ИС включает в себя:

- стадии;

- результаты выполнения работ на каждой стадии (ключевые события — точки завершения работ и принятия решений).

Модель жизненного цикла отражает различные состояния системы, начиная с момента возникновения необходимости в данной ИС и заканчивая моментом ее полного выхода из употребления.

Типы моделей жизненного цикла ИС

В настоящее время известны и используются следующие модели жизненного цикла:

1) Каскадная модель (рис. 1) предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе.

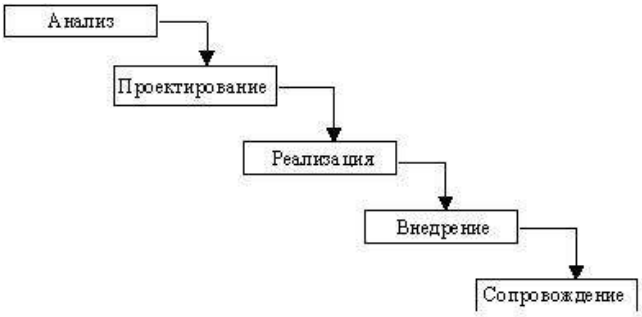

Рис. 1 Каскадная модель жц системы.

2) Поэтапная модель с промежуточным контролем (рис. 2). Разработка ИС ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют учитывать реально существующее взаимовлияние результатов разработки на различных этапах; время жизни каждого из этапов растягивается на весь период разработки.

Рис. 2 Поэтапная модель жц системы.

3) Спиральная модель (рис. 3). На каждом витке спирали выполняется создание очередной версии продукта, уточняются требования проекта, определяется его качество, и планируются работы следующего витка. Особое внимание уделяется начальным этапам разработки - анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования).

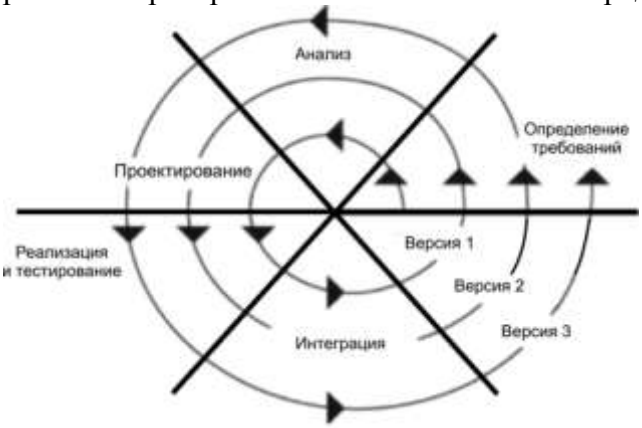

Рис. 3. Спиральная модель жц системы.

Достоинства и недостатки моделей жизненного цикла ИС

Для разработки простых приложений эффективным является каскадный способ. В этом случае информационная система представляет собой единый, функционально и информационно независимый блок. Каждый этап завершается после полного выполнения и документального оформления всех предусмотренных работ. Положительными сторонами применения каскадного подхода являются:

- формирование на каждом этапе законченного набора проектной документации, отвечающий критериям полноты и согласованности;

- выполняемые в логической последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении относительно простых ИС, когда в самом начале разработки можно достаточно точно и полно сформулировать все требования к системе.

Основным недостатком этого подхода является то, что реальный процесс создания системы никогда полностью не укладывается в такую жесткую схему, постоянно возникает потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений.

В результате реальный процесс создания ИС оказывается соответствующим поэтапной модели с промежуточным контролем.

Спиральная модель ЖЦ была предложена для преодоления перечисленных проблем. На этапах анализа и проектирования реализуемость технических решений и степень удовлетворения потребностей заказчика проверяется путем создания прототипов. Каждый виток спирали соответствует созданию работоспособного фрагмента или версии системы. Это позволяет уточнить требования, цели и характеристики проекта, определить качество разработки, спланировать работы следующего витка спирали. Таким образом углубляются и последовательно конкретизируются детали проекта и в результате выбирается обоснованный вариант, который удовлетворяет действительным требованиям заказчика и доводится до реализации. Основная проблема спирального цикла - определение момента перехода на следующий этап. Для ее решения вводятся временные ограничения на каждый из этапов жизненного цикла, и переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. Планирование производится на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Процессы жизненного цикла ИС

Процесс определяется как совокупность взаимосвязанных действий, преобразующих входные данные в выходные. Описание каждого процесса включает в себя перечень решаемых задач, исходных данных и результатов.

Все процессы ЖЦ ПО делятся на три группы:

1. Основные.
2. Вспомогательные процессы жизненного цикла.
3. Организационные процессы жизненного цикла.

1. К основным процессам жизненного цикла ис относятся:

- Приобретение (действия и задачи заказчика, приобретающего ИС)

- Поставка (действия и задачи поставщика, который снабжает заказчика программным продуктом или услугой)

- Разработка (действия и задачи, выполняемые разработчиком: создание ПО, оформление проектной и эксплуатационной документации, подготовка тестовых и учебных материалов и т. д.)

- Эксплуатация (действия и задачи оператора — организации, эксплуатирующей систему)

- Сопровождение (действия и задачи, выполняемые сопровождающей организацией, то есть службой сопровождения). Сопровождение — внесений изменений в ПО в целях исправления ошибок, повышения производительности или адаптации к изменившимся условиям работы или требованиям.

Среди основных процессов жизненного цикла наибольшую важность имеют три: разработка, эксплуатация и сопровождение.

Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами.

1) Разработка информационной системы включает в себя все работы по созданию информационного программного обеспечения и его компонентов в соответствии с заданными требованиями. Разработка информационного программного обеспечения также включает: оформление проектной и эксплуатационной документации; подготовку

материалов, необходимых для тестирования разработанных программных продуктов; разработку материалов, необходимых для обучения персонала. Разработка является одним из важнейших процессов жизненного цикла информационной системы и, как правило, включает в себя стратегическое планирование, анализ, проектирование и реализацию (программирование).

2) Эксплуатация Эксплуатационные работы можно подразделить на подготовительные и основные. К подготовительным относятся: конфигурирование базы данных и рабочих мест пользователей; обеспечение пользователей эксплуатационной документацией; обучение персонала. Основные эксплуатационные работы включают: непосредственно эксплуатацию; локализацию проблем и устранение причин их возникновения; модификацию программного обеспечения; подготовку предложений по совершенствованию системы; развитие и модернизацию системы.

3) Сопровождение. Наличие квалифицированного технического обслуживания на этапе эксплуатации информационной системы является необходимым условием решения поставленных перед ней задач, причем ошибки обслуживающего персонала могут приводить к явным или скрытым финансовым потерям, сопоставимым со стоимостью самой информационной системы. Основными предварительными действиями при подготовке к организации технического обслуживания информационной системы являются: выделение наиболее ответственных узлов системы и определение для них критичности простоя (это позволит выделить наиболее критичные составляющие информационной системы и оптимизировать распределение ресурсов для технического обслуживания); определение задач технического обслуживания и их разделение на внутренние, решаемые силами обслуживающего подразделения, и внешние, решаемые специализированными сервисными организациями (таким образом производится четкое определение круга исполняемых функций и разделение ответственности); проведение анализа имеющихся внутренних и внешних ресурсов, необходимых для организации технического обслуживания в рамках описанных задач и разделения компетенции (основные критерии для анализа: наличие гарантии на оборудование, состояние ремонтного фонда, квалификация персонала); подготовка плана организации технического обслуживания, в котором необходимо определить этапы исполняемых действий, сроки их исполнения, затраты на этапах, ответственность исполнителей.

2. К вспомогательным процессам относятся:
   документирование, управление конфигурацией, обеспечение качества, верификация, аттестация и др.
3. К организационным процессам относятся:
   управление, создание инфраструктуры, усовершенствование, обучение и др.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. Выбрать модель жизненного цикла проекта по своему варианту и обосновать свой выбор.
3. Составить схему модели жц системы по своему заданию.
4. Составить описание **основных** процессов выбранной схемы жизненного цикла.
5. В электронном виде оформите краткий отчет о выполненной работе.
6. Сдайте выполненную работу.

Форма представления результата:

Выполненное задание, отчет о работе.

Критерии оценки:

Отлично - выбранная модель жц системы обоснована на основе анализа проекта. Составленная схема жц правильная и содержит все элементы. Составлено грамотное и полное описание процессов жц выбранной схемы, работа сдана преподавателю.

Хорошо - выбранная модель жц системы обоснована на основе анализа проекта. Составленная схема жц не совсем точная или содержит не все элементы. Составленное описание процессов жц выбранной схемы выполнено с незначительными ошибками, работа сдана преподавателю.

Удовлетворительно - выбранная модель жц системы не обоснована. Составленная схема жц не точная или содержит не все элементы. Составленное описание процессов жц выбранной схемы выполнено со

значительными ошибками, работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Практическая работа № 4 Проектирование и разработка интерфейса пользователя

Цель: ознакомиться с основными принципами Проектирования и разработки интерфейса пользователя проекта; получить практические навыки по проектированию и разработке интерфейса.

Выполнив работу, Вы будете:

уметь:

- проводить анализ предметной области;
- проектировать и разрабатывать систему по заданным требованиям и спецификациям;
- работать с инструментальными средствами обработки информации;
- анализировать задачу и/или проблему и выделять её составные части;
- определять задачи для поиска информации;
- определять необходимые источники информации;

Материальное обеспечение:

Персональный компьютер

Задание:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания в практической части работы
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Краткие теоретические сведения:

Графический интерфейс пользователя (GUI) представляет собой разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на экране, исполнены в виде графических изображений.

Основной особенностью графического интерфейса является то, что пользователь с помощью устройств ввода имеет произвольный ко всем видимым экранным объектам, являющимся элементами интерфейса и осуществляет непосредственное манипулирование ими.

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации.

Полный цикл разработки интерфейса включает следующие этапы:

1. Исследование
2. Создание пользовательского сценария
3. Описание структуры интерфейса
4. Прототипирование интерфейса
5. Определение стилистики
6. Создание дизайн концепции
7. Разработка

1. На этапе исследования проводится сбор информации о продукте, клиенте, его конкурентах или близких аналогах, сбор статистики использования текущего интерфейса, анализ устройств предполагаемой целевой аудитории.

Этот этап помогает понять для кого разрабатывается интерфейс, с какими ограничениями следует его делать (размеры экранов, интерактивность) и т д.

2. Создание пользовательского сценария

На основе предоставленного описания работы интерфейса создается список задач (пользовательских сценариев), которые может выполнять пользователь в рамках интерфейса. Все задачи расписываются по шагам, которые необходимо предпринять для решения задачи.

Составленные списки шагов для каждой задачи помогают понять где путь для решения слишком долог относительно остальных задач. Этап пользовательских сценариев больше всего подходит для сокращения пути решения задач пользователей в рамках интерфейса.

3. Описание структуры интерфейса. Полученный список шагов на предыдущем этапе, ложится в основу структуры интерфейса. Определяется количество экранов, их краткое содержание и положение в общей структуре. Описание структуры выполняется в виде иерархической схемы.

4. Прототипирование интерфейса. В большинстве случаев выполняют два схематичных прототипа: черновой и финальный.

Черновой прототип представляет собой схематичные изображения экранов, связанные между собой. При черновом варианте на схемах изображены зоны и описания этих зон. Например, список новостей или шапка сайта.

Черновой прототип помогает более наглядно понять на сколько объемным будет сайт, как много информации будет на каждом экране, как много нужно кликать, чтобы добраться до нужной страницы.

Финальный прототип, в котором схемы страниц все еще остаются связанными между друг другом, но на страницах уже видны все кнопки, тексты, чекбоксы, формы и прочие элементы.

В прототипах планируется функционал, расположение элементов страниц относительно друг друга без рассмотрения оформления.

5. Определение стилистики. После этапа исследования и параллельно с этапами проектирования идет определение будущей стилистики интерфейса.

Для выбора стилистики готовятся несколько наборов изображений (moodboards). Эти наборы представлены страничками сайтов, иллюстрациями, кнопками, шрифтовыми композициями, связанными между собой стилистически. Один из этих наборов конечном итоге станет основой дизайн концепции.

6. Создание дизайн концепции. Дизайн концепция призвана показать оформление сайта и дать понять будущий вид всего сайта. Обычно концепция представлена 1—3 экранами интерфейса. Если речь идет о сайте, то стараемся показать вид одной и той же страницы для нескольких устройств. Если в интерфейсе предполагается анимация на экране, участвующих в концепции, то показываем и ее.

7. Разработка. Это этап включает в себя оставшиеся работы по созданию, оформлению и тестированию продукта.

Существуют общие принципы создания интерфейса:

1. Естественность (интуитивность). Работа с системой не должна вызывать у пользователя сложностей в поиске необходимых директив (элементов интерфейса) для управления процессом решения поставленной задачи.

2. Непротиворечивость. Если в процессе работы с системой пользователем были использованы некоторые приемы работы с некоторой частью системы, то в другой части системы приемы работы должны быть идентичны. Также работа с системой через интерфейс должна соответствовать установленным, привычным нормам (например, использование клавиши Enter).

3. Неизбыточность. Это означает, что пользователь должен вводить только минимальную информацию для работы или управления системой. Например, пользователь не должен вводить незначимые цифры (00010 вместо 10). Аналогично, нельзя требовать от пользователя ввести информацию, которая была предварительно введена или которая может быть автоматически получена из системы. Желательно использовать значения по умолчанию где только возможно, чтобы минимизировать процесс ввода информации.

4. Непосредственный доступ к системе помощи. В процессе работы необходимо, чтобы система обеспечивала пользователя необходимыми инструкциями. Система помощи отвечает трем основным аспектам - качество и количество обеспечиваемых команд; характер

сообщений об ошибках и подтверждения того, что система делает. Сообщения об ошибках должны быть полезны и понятны пользователю. При этом отвечать на эти вопросы нужно возможно более вежливым и понятным пользователям языком.

5. Гибкость. Интерфейс системы должен обслуживать пользователя с различными уровнями подготовки. Для неопытных пользователей интерфейс может быть организован как иерархическая структура меню, а для опытных пользователей как команды, комбинации нажатий клавиш и параметры.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. Сформулируйте цели разработки системы для своего задания и начальные требования к системе.
3. Определите нужды пользователей системы, и их ожидания
4. Создайте прототип интерфейса с представлением полного перечня разделов и/или всех страниц, имеющихся в системе.
5. В электронном виде оформите краткий отчет о выполненной работе.
6. Сдайте выполненную работу.

Форма представления результата:

Выполненное задание, схема прототипа, отчет о работе.

Критерии оценки:

Отлично – работа выполнена полностью, схема системы разработана с учетом целей и требований к системе, учтены требования пользователей и основные принципы проектирования графического интерфейса. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Хорошо - работа выполнена полностью, схема системы разработана с учетом целей и требований к системе, но с незначительными ошибками, не до конца учтены требования пользователей и основные принципы проектирования графического интерфейса. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Удовлетворительно - работа выполнена не полностью, схема системы разработана с существенными ошибками, не до конца учтены требования пользователей и основные принципы проектирования графического интерфейса. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 6 Установка и настройка системы контроля версий с разграничением ролей

Цель: получение первоначальных навыков использования систем контроля версий исходного кода программ.

Выполнив работу, Вы будете:

уметь:

- проектировать и разрабатывать систему по заданным требованиям и спецификациям;
- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
- читать, понимать и находить необходимые технические данные и инструкции в руководствах в любом доступном формате;

Материальное обеспечение:

Персональный компьютер, Git

Задание:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания в практической части работы
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Краткие теоретические сведения:

Система контроля версий представляет собой специальное программное обеспечение для совместной работы с постоянно изменяющейся информацией. Она является надстройкой над файловой системой, которая позволяет хранить несколько версий одного и того же файла

Системы контроля версий нужны для:

- Хранения полной истории изменений
- Описания причин всех производимых изменений
- Отмены изменений, если что-то было сделано неправильно
- Поиска причин и ответственного за появление ошибки в программе
- Совместной работы нескольких разработчиков над одним проектом
- Возможности вносить изменения, не мешая работе других разработчиков

Существуют следующие разновидности систем:

1. Централизованные системы контроля версий

Характеризуются следующими признаками:

- Единственное хранилище (репозиторий) для хранения всех файлов проекта
- Каждый пользователь копирует необходимые ему файлы из этого хранилища, изменяет и, затем, загружает измененные файлы обратно в хранилище

2. Распределенные системы контроля версий.

- У каждого пользователя свое хранилище (возможно, не одно)
- Система обеспечивает возможность работы с любыми хранилищами (локальными и удаленными)

Git - одна из систем контроля версий. Предназначена, в основном, для работы распределенной команды разработчиков. Система Git очень экономична и не требует рассылки большого количества файлов. Отслеживаются и пересылаются изменения в файлах и ссылки на эти изменения. То есть основная рассылка это рассылка разницы в ваших редактированиях.Отсылаются только различия в папках и файлах. В любой момент времени вы можете возвратиться к тому или иному состоянию системы. Многие компании уделяют внимание хорошей и быстрой коммуникации между сотрудниками. В этом отношении, система контроля версий предоставляет большие возможности.

Основная задача системы управление версий — это упрощение работы с потоками изменяющейся информации. Главной парадигмой системы управления версий является локализация данных каждого разработчика проекта. Каждый разработчик имеет на своей машине локальный репозиторий. В случае необходимости изменения отправляются из локального репозитория в удаленное хранилище в определенную ветку. И любой разработчик из распределенной команды может скачать новые изменения в проекте, чтобы продолжить совместную работу над проектом.

В стандартной поставке Git поддерживается взаимодействие с CVS (импорт и экспорт, эмуляция CVS-сервера) и Subversion (частичная поддержка импорта и экспорта). Стандартный инструмент импорта и экспорта внутри экосистемы — архивы серий версионированных файлов в форматах .tar.gz и .tar.bz2.

Рабочее дерево (рабочая директория, рабочее пространство) – это дерево исходных файлов, которые вы можете видеть и редактировать.

Индекс (область подготовленных файлов, staging area) – это один большой бинарный файл .git/index, в котором указаны все файлы текущей ветки, их SHA1, временные метки и имена. Это не отдельная директория с копиями файлов.

Указатель HEAD – это ссылка на последний коммит в текущей извлеченной ветке.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. Установите Git с официального сайта.
3. Настройка пользователя и емейл:

   git config --global user.name "My Name"

   git config --global user.email myEmail@example.com

4. Создание нового репозитория. Git хранит свои файлы и историю прямо в папке проекта. Чтобы создать новый репозиторий, нам нужно открыть терминал, зайти в папку нашего проекта и выполнить команду init. Это включит приложение в этой конкретной папке и создаст скрытую директорию .git, где будет храниться история репозитория и настройки. Создайте на рабочем столе папку под названием git_exercise. Для этого в окне терминала введите:

   $ mkdir Desktop/git_exercise/

   $ cd Desktop/git_exercise/

   $ git init

   Командная строка должна вернуть что-то вроде:

   Initialized empty Git repository in /home/user/Desktop/git_exercise/.git/

   Это значит, что наш репозиторий был успешно создан, но пока что пуст. Теперь создайте текстовый файл под названием hello.txt и сохраните его в директории git_exercise.

   Определение состояния. status — это команда, которая показывает информацию о текущем состоянии репозитория: актуальна ли информация на нём, нет ли чего-то нового, что поменялось, и так далее. Запуск git status на нашем свежесозданном репозитории:

   $ git status

   On branch master

   Initial commit

   Untracked files:

   (use "git add ..." to include in what will be committed)

   hello.txt

   Сообщение говорит о том, что файл hello.txt неотслеживаемый. Для того, чтобы начать отслеживать новый файл, нужно его специальным образом объявить.

5. Подготовка файлов

   В git есть концепция области подготовленных файлов. В эту область можно добавить файлы (или части файлов, или даже одиночные строчки) командой add, добавить все нужное

в репозиторий (создаем слепок нужного нам состояния) командой commit.В нашем случае у нас только один файл, так что добавим его:

$ git add hello.txt

Если нам нужно добавить все, что находится в директории, мы можем использовать

$ git add -A

Проверим статус снова, на этот раз мы должны получить другой ответ:

$ git status

On branch master

Initial commit

Changes to be committed:

(use "git rm --cached ..." to unstage)

new file: hello.txt

Файл готов к коммиту. Сообщение о состоянии также говорит нам о том, какие изменения относительно файла были проведены в области подготовки — в данном случае это новый файл, но файлы могут быть модифицированы или удалены.

Коммит(фиксация изменений). Коммит представляет собой состояние репозитория в определенный момент времени. Мы можем вернуться и увидеть состояние объектов на определенный момент времени.

Чтобы зафиксировать изменения, нам нужно хотя бы одно изменение в области подготовки (мы только что создали его при помощи git add), после которого мы может коммитить:

$ git commit -m "Initial commit."

Эта команда создаст новый коммит со всеми изменениями из области подготовки (добавление файла hello.txt). Ключ -m и сообщение «Initial commit.» — это созданное пользователем описание всех изменений, включенных в коммит. Считается хорошей практикой делать коммиты часто и всегда писать содержательные комментарии.

6. Удаленные репозитории

Сейчас наш коммит является локальным — существует только в директории .git на нашей файловой системе.

Подключение к удаленному репозиторию

Чтобы загрузить что-нибудь в удаленный репозиторий, сначала нужно к нему подключиться. Можно использовать адрес https://github.com/tutorialzine/awesome-project, но также можно попробовать создать свой репозиторий в GitHub, BitBucket или любом другом сервисе. Регистрация и установка может занять время, но все подобные сервисы предоставляют хорошую документацию.

Чтобы связать наш локальный репозиторий с репозиторием на GitHub, выполним следующую команду в терминале. Обратите внимание, что нужно обязательно изменить URI репозитория на свой.

This is only an example. Replace the URI with your own repository address.

$ git remote add origin https://github.com/tutorialzine/awesome-project.git

Проект может иметь несколько удаленных репозиториев одновременно. Чтобы их различать, мы дадим им разные имена. Обычно главный репозиторий называется origin.

Отправка изменений на сервер . Этот процесс происходит каждый раз, когда мы хотим обновить данные в удаленном репозитории. Команда, предназначенная для этого - push. Она принимает два параметра: имя удаленного репозитория (мы назвали наш origin) и ветку, в которую необходимо внести изменения (master — это ветка по умолчанию для всех репозиториев).

$ git push origin master

Counting objects: 3, done.

Writing objects: 100% (3/3), 212 bytes | 0 bytes/s, done.

Total 3 (delta 0), reused 0 (delta 0)

To https://github.com/tutorialzine/awesome-project.git

* [new branch] master -> master

В зависимости от сервиса, который вы используете, вам может потребоваться аутентифицироваться, чтобы изменения отправились. Если все сделано правильно, то когда вы посмотрите в удаленный репозиторий при помощи браузера, вы увидите файл hello.txt

7. Клонирование репозитория. Сейчас другие пользователи GitHub могут просматривать ваш репозиторий. Они могут скачать из него данные и получить полностью работоспособную копию вашего проекта при помощи команды clone.

$ git clone https://github.com/tutorialzine/awesome-project.git

Новый локальный репозиторий создается автоматически с GitHub в качестве удаленного репозитория.

Запрос изменений с сервера. Если вы сделали изменения в вашем репозитории, другие пользователи могут скачать изменения при помощи команды pull.

$ git pull origin master

From https://github.com/tutorialzine/awesome-project

* branch master -> FETCH_HEAD

Already up-to-date.

Так как новых коммитов с тех пор, как мы склонировали себе проект, не было, никаких изменений доступных для скачивания нет.

8. Ветвление. Во время разработки новой функциональности считается хорошей практикой работать с копией оригинального проекта, которую называют веткой. Ветви имеют свою собственную историю и изолированные друг от друга изменения до тех пор, пока вы не решаете слить изменения вместе. Это происходит по набору причин:

- Уже рабочая, стабильная версия кода сохраняется.
- Различные новые функции могут разрабатываться параллельно разными программистами.
- Разработчики могут работать с собственными ветками без риска, что кодовая база поменяется из-за чужих изменений.
- В случае сомнений, различные реализации одной и той же идеи могут быть разработаны в разных ветках и затем сравниваться.

1. Создание новой ветки

Основная ветка в каждом репозитории называется master. Чтобы создать еще одну ветку, используем команду branch <name>

$ git branch amazing_new_feature

Это создаст новую ветку, пока что точную копию ветки master.

2. Переключение между ветками

Сейчас, если мы запустим branch, мы увидим две доступные опции:

$ git branch

amazing_new_feature

* master

master — это активная ветка, она помечена звездочкой. Для переключения воспользуемся командой checkout, она принимает один параметр — имя ветки, на которую необходимо переключиться.

$ git checkout amazing_new_feature

3. Слияние веток

Создадим feature.txt, добавим и закоммитим:

$ git add feature.txt

$ git commit -m "New feature complete.”

Изменения завершены, теперь мы можем переключиться обратно на ветку master.

$ git checkout master

Теперь, если мы откроем наш проект в файловом менеджере, мы не увидим файла feature.txt, потому что мы переключились обратно на ветку master, в которой такого файла не существует. Чтобы он появился, нужно воспользоваться merge для объединения веток (применения изменений из ветки amazing_new_feature к основной версии проекта).

$ git merge amazing_new_feature

Теперь ветка master актуальна. Ветка amazing_new_feature больше не нужна, и ее можно удалить.

$ git branch -d awesome_new_feature

9. Отслеживание изменений, сделанных в коммитах

У каждого коммита есть свой уникальный идентификатор в виде строки цифр и букв. Чтобы просмотреть список всех коммитов и их идентификаторов, можно использовать команду log:

[spoiler title='Вывод git log']
$ git log
commit ba25c0ff30e1b2f0259157b42b9f8f5d174d80d7
Author: Tutorialzine
Date: Mon May 30 17:15:28 2016 +0300
New feature complete
commit b10cc1238e355c02a044ef9f9860811ff605c9b4
Author: Tutorialzine
Date: Mon May 30 16:30:04 2016 +0300
Added content to hello.txt
commit 09bd8cc171d7084e78e4d118a2346b7487dca059
Author: Tutorialzine
Date: Sat May 28 17:52:14 2016 +0300
Initial commit
[/spoiler]

Как вы можете заметить, идентификаторы довольно длинные, но для работы с ними не обязательно копировать их целиком — первых нескольких символов будет вполне достаточно. Чтобы посмотреть, что нового появилось в коммите, мы можем воспользоваться командой

show [commit][spoiler title='Вывод git show']
$ git show b10cc123
commit b10cc1238e355c02a044ef9f9860811ff605c9b4
Author: Tutorialzine
Date: Mon May 30 16:30:04 2016 +0300
Added content to hello.txt
diff --git a/hello.txt b/hello.txt
index e69de29..b546a21 100644
--- a/hello.txt
+++ b/hello.txt
@@ -0,0 +1 @@
+Nice weather today, isn't it?
[/spoiler]

Чтобы увидеть разницу между двумя коммитами, используется команда diff (с указанием промежутка между коммитами):

[spoiler title='Вывод git diff']
$ git diff 09bd8cc..ba25c0ff
diff --git a/feature.txt b/feature.txt
new file mode 100644
index 0000000..e69de29

diff --git a/hello.txt b/hello.txt
index e69de29..b546a21 100644
--- a/hello.txt
+++ b/hello.txt
@@ -0,0 +1 @@
+Nice weather today, isn't it?
[/spoiler]

Мы сравнили первый коммит с последним, чтобы увидеть все изменения, которые были когда-либо сделаны. Обычно проще использовать git difftool, так как эта команда запускает графический клиент, в котором наглядно сопоставляет все изменения.

10. Возвращение файла к предыдущему состоянию

Это делается уже знакомой нам командой checkout, которую мы ранее использовали для переключения между ветками. Но она также может быть использована для переключения между коммитами В следующем примере мы возьмем файл hello.txt и откатим все изменения, совершенные над ним к первому коммиту. Чтобы сделать это, мы подставим в команду идентификатор нужного коммита, а также путь до файла:

$ git checkout 09bd8cc1 hello.txt

11. Исправление коммита

Если вы опечатались в комментарии или забыли добавить файл и заметили это сразу после того, как закоммитили изменения, вы легко можете это поправить при помощи commit —amend. Эта команда добавит все из последнего коммита в область подготовленных файлов и попытается сделать новый коммит. Это дает вам возможность поправить комментарий или добавить недостающие файлы в область подготовленных файлов.

Для более сложных исправлений, например, не в последнем коммите или если вы успели отправить изменения на сервер, нужно использовать revert. Эта команда создаст коммит, отменяющий изменения, совершенные в коммите с заданным идентификатором.

Самый последний коммит может быть доступен по алиасу HEAD:

$ git revert HEAD

Для остальных будем использовать идентификаторы:

$ git revert b10cc123

При отмене старых коммитов нужно быть готовым к тому, что возникнут конфликты. Такое случается, если файл был изменен еще одним, более новым коммитом. И теперь git не может найти строчки, состояние которых нужно откатить, так как они больше не существуют.

12. Разрешение конфликтов при слиянии

Помимо сценария, описанного в предыдущем пункте, конфликты регулярно возникают при слиянии ветвей или при отправке чужого кода. Иногда конфликты исправляются автоматически, но обычно с этим приходится разбираться вручную — решать, какой код остается, а какой нужно удалить.

Давайте посмотрим на примеры, где мы попытаемся слить две ветки под названием john_branch и tim_branch. И Тим, и Джон правят один и тот же файл: функцию, которая отображает элементы массива.Джон использует цикл:

```
// Use a for loop to console.log contents.
for(var i=0; i<arr.length; i++) {
console.log(arr[i]);
}
```

Тим предпочитает forEach:

```
// Use forEach to console.log contents.
arr.forEach(function(item) {
console.log(item);
});
```

Они оба коммитят свой код в соответствующую ветку. Теперь, если они попытаются слить две ветки, они получат сообщение об ошибке:

$ git merge tim_branch

Auto-merging print_array.js

CONFLICT (content): Merge conflict in print_array.js

Automatic merge failed; fix conflicts and then commit the result.

Система не смогла разрешить конфликт автоматически, значит, это придется сделать разработчикам. Приложение отметило строки, содержащие конфликт:

[spoiler title='Вывод']

<<<<<<< HEAD // Use a for loop to console.log contents. for(var i=0; i<arr.length; i++) { console.log(arr[i]); } ======= // Use forEach to console.log contents. arr.forEach(function(item) { console.log(item); }); >>>>>>> Tim's commit.

[/spoiler]

Над разделителем ======= мы видим последний (HEAD) коммит, а под ним - конфликтующий. Таким образом, мы можем увидеть, чем они отличаются и решать, какая версия лучше. Или вовсе написать новую. В этой ситуации мы так и поступим, перепишем все, удалив разделители, и дадим git понять, что закончили.

// Not using for loop or forEach.

// Use Array.toString() to console.log contents.

console.log(arr.toString());

Когда все готово, нужно закоммитить изменения, чтобы закончить процесс:

$ git add -A

$ git commit -m "Array printing conflict resolved."

Как вы можете заметить, процесс довольно утомительный и может быть очень сложным в больших проектах. Многие разработчики предпочитают использовать для разрешения конфликтов клиенты с графическим интерфейсом. (Для запуска нужно набрать git mergetool).

13. Настройка .gitignore

В большинстве проектов есть файлы или целые директории, в которые мы не хотим (и, скорее всего, не захотим) коммитить. Мы можем удостовериться, что они случайно не попадут в git add -A при помощи файла .gitignore

- Создайте вручную файл под названием .gitignore и сохраните его в директорию проекта.
- Внутри файла перечислите названия файлов/папок, которые нужно игнорировать, каждый с новой строки.
- Файл .gitignore должен быть добавлен, закоммичен и отправлен на сервер, как любой другой файл в проекте.

14. В электронном виде оформите краткий отчет о выполненной работе.

15. Сдайте выполненную работу.

Форма представления результата:

Выполненное задание, отчет о работе.

Критерии оценки:

Отлично – работа выполнена полностью, без ошибок, продемонстрирован результат настройки системы, выполнен отчет о работе, работа сдана преподавателю.

Хорошо - работа выполнена не полностью, или с незначительными ошибками, продемонстрирован результат настройки системы, выполнен отчет о работе, работа сдана преподавателю.

Удовлетворительно - работа выполнена не полностью, или со значительными ошибками, продемонстрирован результат настройки системы, отчет о работе выполнен не до конца, работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 7 Разработка графического интерфейса пользователя

Цель: закрепление практических навыков разработки графического интерфейса пользователя.

Выполнив работу, Вы будете:

уметь:

- проектировать и разрабатывать систему по заданным требованиям и спецификациям;
- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
- читать, понимать и находить необходимые технические данные и инструкции в руководствах в любом доступном формате;
-решать прикладные вопросы программирования и языка сценариев для создания программ;

Материальное обеспечение:

Персональный компьютер, Sublime Text 3,

Задание:

1. Ознакомиться с теоретическим материалом.
2. Выполнить задания в практической части работы
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Краткие теоретические сведения:

Графический интерфейс пользователя (GUI) представляет собой разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на экране, исполнены в виде графических изображений.

Основной особенностью графического интерфейса является то, что пользователь с помощью устройств ввода имеет произвольный ко всем видимым экранным объектам, являющимся элементами интерфейса и осуществляет непосредственное манипулирование ими.

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации.

Можно выделить следующие виды графического интерфейса пользователя:

- простой: типовые экранные формы и стандартные элементы интерфейса, обеспечиваемые самой подсистемой;
- истинно графические, двухмерные: нестандартные элементы интерфейса и оригинальные метафоры, реализованные собственными средствами приложения или сторонней библиотекой;
- трёхмерный.

Проектирование графического интерфейса, как правило осуществляется командой, в которой участники обладают необходимыми компетенциями: навыками художника или графика, специалиста по анализу требований, системного проектировщика, программиста, эксперта по технологии и других.

Основное преимущество графического интерфейса заключается в том, что он является интуитивно понятным и удобным в использовании, за счет чего достигается максимально эффективное взаимодействие между программной системой и пользователем.

Для достижения этой эффективности необходимо решить ряд вопросов:

1. Определение задачи или набора задач, для которых предназначена система.

2. Определить потребности пользователя системы, учесть привычные способы работы пользователя. Выделить возможные трудности, с которыми может столкнуться пользователь при работе в системе.

3. Определить наименьший набор возможного количества действий со стороны пользователя для реализации поставленных перед ним задач.

Одним из главных принципов при проектировании является создание максимально простого и понятного интерфейса.

Для этого в дизайне интерфейса требуется учитывать различные аспекты: от цвета, формы, пропорций элементов до особенностей восприятия их человеком.

Среди наиболее важных можно отметить следующие:

1. Цвет. Восприятие цвета является субъективным, и может меняться в зависимости от ситуации даже для одного человека. Однако считается, что в интерфейсе лучше воспринимаются теплые цвета.
2. Форма. Восприятие формы, так же, как и цвета достаточно субъективно. Однако в большинстве случаев лучше применять мягкие, скругленные формы.
3. Часто используемые элементы должны быть выделены, например, размером или цветом.
4. Иконки в программе должны быть очевидными или подписанными
5. Располагать элементы интерфейса на экране нужно по принципу «Золотого сечения». Весь отрезок относится к большей его части так, как большая часть, относится к меньшей. Например, общая ширина сайта 900px, делим 900 на 1.62, получаем ~555px, это ширина блока с контентом. Теперь от 900 отнимаем 555 и получаем 345px. Это ширина меньшей части.
6. Экранные элементы группируют по значимости. Обычно это осуществляют по правилам чтения текста: слева направо, сверху вниз. В случае с сенсорными экранами, самые важные элементы, располагаются в области действия больших пальцев рук.
7. Необходимо учитывать привычки пользователя, т.е. интерфейс должен иметь как можно больше аналогий, с известными пользователю вещами.
8. Размещать элементы стоит ближе к той части, где большую часть времени находится курсор пользователя.
9. Элемент интерфейса можно считать видимым, если он либо в данный момент доступен для органов восприятия человека, либо он был настолько недавно воспринят, что еще не успел выйти из кратковременной памяти. Для нормальной работы интерфейса, должны быть видимы только необходимые вещи — те, что идентифицируют части работающих систем, и те, что отображают способ, которым пользователь может взаимодействовать с устройством.
10. Отступы между элементами лучше делать равными или кратными друг-другу.

Порядок выполнения работы:

1. Изучить теоретическую информацию.
2. Используя прототип графического интерфейса из практической работы 4 разработать макет графического интерфейса по своему заданию.
3. На основе созданного макета разработать интерфейс по заданию.
4. Составить краткий отчёт, содержащий краткое описание процесса создания и компиляции проекта.

Форма представления результата:

Выполненное задание, макет, отчет о работе.

Критерии оценки:

Отлично – работа выполнена полностью, макеты системы разработан на основе прототипа, с учетом целей и требований к системе, учтены требования пользователей и

основные принципы проектирования графического интерфейса. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Хорошо - работа выполнена полностью, макет системы разработан на основе прототипа, с незначительными ошибками, не до конца учтены требования пользователей или основные принципы проектирования графического интерфейса. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Удовлетворительно - работа выполнена не полностью, макеты системы не соответствует прототипу или разработан с существенными ошибками, не до конца учтены требования пользователей и основные принципы проектирования графического интерфейса. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 8 Реализация алгоритмов обработки числовых данных. Отладка приложения

Цель: закрепление практических навыков разработки интерфейса пользователя, закрепление навыков отладки приложений.

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
-решать прикладные вопросы программирования и языка сценариев для создания программ;
- создавать и управлять проектом по разработке приложения;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Разработать приложение калькулятор
2. Выполнить отладку приложения
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Порядок выполнения работы:

1. Создать макет приложения калькулятор, который должен содержать:
- поле ввода
- цифровые кнопки
- знаки математических операций

2. Разработать на javascript приложение, реализующее стандартные операции калькулятора. При реализации необходимо учитывать:
- проверку вводимых значений
- ввод вещественных чисел
- обработку возможных ошибок, например, деления на ноль
- возможность очистки поля ввода

3. Осуществить отладку приложения.
4. Составить краткий отчёт, содержащий описание процесса создания и отладки проекта.

Форма представления результата:

Выполненное задание, макет, отчет о работе.

Критерии оценки:

Отлично - работа выполнена полностью, все необходимые функции реализованы. Приложение разработано с учетом общепринятых стандартов GUI. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Хорошо - работа выполнена не полностью, реализованы большинство функций. Приложение разработано с учетом общепринятых стандартов GUI. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Удовлетворительно - работа выполнена частично, реализованы мало функций. Приложение разработано с учетом общепринятых стандартов GUI. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 9 Реализация алгоритмов поиска. Отладка приложения поиск

Цель: закрепление практических навыков разработки пользовательских приложений, работа по созданию механизмов поисковой строки для приложения, закрепление навыков отладки приложений.

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
-решать прикладные вопросы программирования и языка сценариев для создания программ;
- создавать и управлять проектом по разработке приложения;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Ознакомиться с теоретическим материалом.
2. Разработать и реализовать в своем приложении механизм поиска.
3. Выполнить отладку приложения
4. Составить в электронном виде отчет о работе.
5. Сдать выполненную работу.

Краткие теоретические сведения:

Поиск по странице, может понадобиться для страниц предоставляющих большой объем данных. Удобнее осуществлять если для этого реализованы отдельные элементы поиска, например стандартная строка для ввода искомых значений.

При реализации механизма поиска необходимо соблюдать некоторые правила:

1. Необходимо проверять корректность ввода данных, например, удаление лишних пробелов по бокам фразы для поиска.
2. Возможность выделения всех найденных элементов, например, цветом.
3. Корректный вывод результатов поиска, например, сообщения, об отсутствии совпадений.

Порядок выполнения работы:

1. Разработать и создать в своем приложении механизм текстового поиска, позволяющий увидеть все найденные на странице соответствия. Осуществлять проверку вводимых значений.
2. Осуществить отладку приложения.
3. Составить краткий отчёт, содержащий описание процесса создания и отладки проекта.

Форма представления результата:

Выполненное задание, макет, отчет о работе.

Критерии оценки:

Отлично - работа выполнена полностью, реализован механизм поиска. Приложение разработано с учетом общепринятых стандартов GUI. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Хорошо - работа выполнена полностью, но не реализованы все функции или нарушены общепринятые стандарты GUI. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Удовлетворительно - работа выполнена частично, нарушены общепринятые стандарты GUI. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 10 Реализация обработки табличных данных. Отладка приложения

Цель: закрепление практических навыков разработки пользовательских приложений с использованием табличных данных, закрепление навыков отладки приложений.

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
-решать прикладные вопросы программирования и языка сценариев для создания программ;
- создавать и управлять проектом по разработке приложения;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Создать на своей странице таблицу по заданию
2. Выполнить отладку приложения
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Порядок выполнения работы:

1. Добавить свое приложение таблицу по заданию. Формат таблицы должен соответствовать странице, выделить заголовки строк и столбцов, итоговые значения.
2. Предусмотреть возможность фильтрации таблицы по столбцам.
3. Осуществить отладку приложения.
4. Составить краткий отчёт, содержащий описание процесса создания и отладки проекта.

Форма представления результата:

Выполненное задание, макет, отчет о работе.

Критерии оценки:

Отлично - создана и добавлена в проект таблица, она соответствует заданию. Оформление выдержано, выполняется фильтрация данных. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Хорошо - создана и добавлена в проект таблица, она частично не соответствует заданию. Оформление выдержано частично, выполняется фильтрация данных. Составлен отчет о выполненной работе.

Удовлетворительно - создана и добавлена в проект таблица, не полностью соответствует заданию, ошибки в оформлении. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 11 Разработка и отладка генератора случайных символов

Цель: научиться генерировать строки случайных символов средствами javascript

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
-решать прикладные вопросы программирования и языка сценариев для создания программ;
- создавать и управлять проектом по разработке приложения;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Разработать генератор последовательности случайных символов
2. Выполнить отладку приложения
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Порядок выполнения работы:

1. Разработать алгоритм и создать генератор последовательности случайных символов.
2. На странице создать элемент для генерирования строки из случайных символов. В качестве символов могут выступать буквы русского и латинского, цифры, знаки арифметических операций. Предусмотреть генерацию букв в разных регистрах.
3. Осуществить отладку приложения.
4. Составить краткий отчёт, содержащий описание процесса создания и отладки проекта.

Форма представления результата:

Выполненное задание, макет, отчет о работе.

Критерии оценки:

Отлично - создан и отлажен генератор случайных символов с соблюдением всех условий. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Хорошо - создан и отлажен генератор случайных символов с небольшими неточностями. Составлен отчет о выполненной работе. Работа сдана преподавателю.

Удовлетворительно – работа выполнена без учета условий задания. Составленый отчет о выполненной работе содержит ошибки. Работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 12 Разработка приложений для моделирования процессов и явлений. Отладка приложения

Цель: закрепление практических навыков разработки и отладки пользовательских приложений для решения практических задач.

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
-решать прикладные вопросы программирования и языка сценариев для создания программ;
- создавать и управлять проектом по разработке приложения;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;
- определять задачи для поиска информации;
- читать, понимать и находить необходимые технические данные и инструкции в руководствах в любом доступном формате;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. По индивидуальному заданию разработать приложение, выполнить его отладку и тестирование.
2. Составить в электронном виде отчет о работе.
3. Сдать выполненную работу.

Порядок выполнения работы:

1. Рассмотреть индивидуальное задание, определить цели создания приложения, пользовательскую аудиторию.
2. Разработать алгоритм реализации для решения задачи.
3. Создать макет приложения.
4. Разработать приложение для решения поставленной задачи.
5. Осуществить отладку и тестирование приложения для различных вариантов входных параметров.
6. Составить краткий отчёт, содержащий описание процесса создания и отладки проекта.

Варианты заданий:

1. Создание простого текстового редактора для создания и сохранения заметок. Редактор должен реализовывать следующие возможности: сохранение; редактирование; удаление; выделение цветом; при закрытии окна / браузера данные должны сохраняться, а при «возвращении» пользователя – динамически подгружаться.
2. Создание теста – приложения для тестирования знаний в какой-либо предметной области. Приложение должно содержать возможность проходить тест несколько раз, ограничение по времени; отображать общее количество вопросов и количество заданных вопросов; должны отображаться результаты прохождения теста: общее время тестирования, количество правильных ответов, резюме тестирования).
3. Создание приложения, реализующего функции инженерного калькулятора. В приложении должны быть реализованы вычисления для отрицательных и дробных

чисел, тригонометрических функций, операции возведения числа в конкретную степень, проверки вводимых значений и обработки ошибок при вводе.

4. Создание приложения, реализующего функции калькулятора для перевода целых и дробных чисел в системах счисления и арифметических операций с числами. В приложении должны быть реализованы возможность переключения между системами счисления, проверка правильности вводимых значений, обработка ошибок.
5. Создание игры «мемо».

   Описание игры:

   - игра ведется на время;
   - после запуска приложения перед игроком открывается поле размером n X n карточек, изначально карты лежат лицом вверх;
   - через m секунд карты поворачиваются лицевой стороной вниз; запускается таймер игры;
   - после клика по карточке, появляется изображение;
   - необходимо найти такую же картинку среди закрытых карт;
   - открытая карточка будет отображаться до тех пор, пока пользователь не нажмет на вторую;
   - если нашлись две одинаковые карточки, то они исчезают из списка, а если нет – переворачиваются в исходное положение;
   - когда нашлись все совпадения – пользователю выводится сообщение;
   - игра может закончиться в 2-х случаях: если вышло время,, определенное на игру или игрок правильно составил все пары карт;
   - в результате должна выводиться пользовательская статистика (количество выигрышей/проигрышей, лучшее время).
6. Создание игры «крестики-нолики».

   Описание игры:

   - игра ведется на время;
   - после запуска приложения перед игроком открывается поле размером n X n;
   - в поле случайным образом генерируется число, показывающее, кто осуществляет первый ход;
   - игроки поочередно выставляют знаки «0» или «X» до тех пор, пока не закончится время игры или не будет составлена выигрышная комбинация;
   - в результате должна выводиться информация о результатах игры, время, проигравший и выигравший игрок
7. Создание приложения «органайзер». Приложение должно реализовывать следующие функции: осуществлять ввод задачи в специальное поле; сохранение и появление в списке новой задачи по нажатию на кнопку; возможность редактировать любой пункт; просмотр выполненных и активных задач, даты их создания; при закрытии окна/браузера данные должны сохраняться, а при «возвращении» пользователя динамически подгружены.
8. Создание приложения по отслеживанию расходов. Приложение должно реализовывать следующие функции: добавление исходных данных по имеющимся суммам на расходы, разбиение исходных сумм на категории трат, добавление и разбивки расходов по категориям, генерацию итоговых отчетов, основанный на входящих данных, и пользовательские уведомления.
9. Создание приложения по матричной арифметике. возможность ввода размерности матриц; формирование исходных данных для матриц путем ввода значений или случайным образом из конкретного диапазона; кнопки для реализации матричных вычислений: сложения, вычитание, умножения, получения обратной матрицы, вычисления определителя; должна быть предусмотрена проверка правильности вводимых значений и действий.

10. Создание приложения «Расписание приема врача». Приложение должно реализовывать следующие функции: возможность осуществить запись на прием ко врачу на конкретное число в свободное время. В таблице должно отображаться текущее расписание с указанием занятого и свободного времени. По итогам записи должно быть сформировано уведомление о дате, времени, месте приема у конкретного врача.
11. Создание приложения «Викторина» Приложение должно реализовывать следующие функции выбор случайным образом различных вопросов из базы; возможность выбора варианта ответа; подсчет количества правильных результатов, итоговую информацию с общим результатом и правильными вариантами ответов.
12. Создание приложения «Адресная книга». Приложение должно реализовывать следующие функции: добавление, редактирование, удаление контактов, включая номера, адреса электронной почты и небольшие заметки о них; возможность разбивать контакты на группы, создавать приоритетные группы или контакты.
13. Создание приложения «Калькулятор для ипотеки». Приложение должно реализовывать следующие функции: вычисление месячных выплат в течении указанного периода с заданной процентной ставкой; приложение должно осуществлять выбор из нескольких возможных предложений банков.
14. . Создание приложения «Поиск книг». Приложение должно реализовывать следующие функции: поиска книги по запросу (название, автор и т. д.); возможность осуществлять поиск по одному или нескольким параметрам; отображение найденного списка книг со всей информацией о них.
15. Создание приложения «Калькулятор» для римских цифр. В приложении должны быть реализованы возможности ввода чисел римскими цифрами; арифметические действия с ними; вывод итогового результата; обработка правильности вводимых значений и вычислений.
16. Создание приложения «Шифр Цезаря». В приложении должны быть реализованы возможности вывода таблицы всех возможных в «шифре Цезаря» преобразований введенной текстовой строки; неалфавитные символы — знаки препинания, пробелы, цифры в шифре не меняются.
17. Создание приложения «Игра в пятнашки».
    Описание игры:
    - после запуска приложения перед игроком открывается поле размером 4 X 4 с произвольно размещенными числами от1 до 15;
    - одна позиция на поле остается незанятой для передвижения чисел;
    - задача игрока перемещая числа при помощи незанятой позиции, выставить все числа в порядке возрастания;
    - в результате должна выводиться информация о количестве выполненных шагов, времени игры
18. Создание приложения «Игра в слова».
    Описание игры:
    - после запуска приложения включается таймер; перед игроком открывается поле размером n X n с произвольно размещенными буквами (каждая буква является частью слова, т е на поле размещены несколько слов, путем перемешивания букв);
    - игроку необходимо за ограниченное время найти все слова, нажимая подряд буквы слова;
    - когда слово угадано, оно подсвечивается или убирается их поля;
    - игра заканчивается, когда угаданы все слова или вышло отведённое на игру время;
    - в результате должна выводиться информация о результате игры, список всех найденных слов;

19. Создание приложения для контроля за спортивными достижениями. Приложение должно отслеживать динамику ежедневных спортивных занятий пользователя. В приложении должны быть реализованы возможности ввода программы занятий на конкретный срок (месяц, неделя и т д); фиксация результатов, анализ результатов за указанный период, в виде графика или диаграммы.
20. Создание справочного приложения для школьников по геометрии. В приложении содержится справочная о геометрических фигурах, формулы для расчетов; есть возможность выбора вида фигуры, формулы, ввода исходных данных.

Форма представления результата:

Выполненное задание, отчет о работе.

Критерии оценки:

Отлично - Правильно разработан алгоритм реализации для решения задачи, макет приложения соответствует основным требованиям проектирования приложений, приложение разработано согласно макету, приложение решает поставленную задачу, выполнена отладка и тестирование приложения для различных вариантов входных параметров. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю.

Хорошо - Разработанный алгоритм реализации решения задачи имеет незначительные ошибки, макет приложения соответствует основным требованиям проектирования приложений, приложение разработано согласно макету, приложение решает поставленную задачу не совсем точно, выполнена отладка и тестирование приложения для различных вариантов входных параметров. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю.

Удовлетворительно - Разработанный алгоритм реализации решения задачи имеет значительные ошибки или не до конца реализует решение задачи, макет приложения плохо соответствует основным требованиям проектирования приложений, приложение разработано согласно макету, приложение решает поставленную задачу не полностью или не точно, выполнена отладка и тестирование приложения для различных вариантов входных параметров. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 13 Интеграция модуля в информационную систему

Цель: закрепление практических навыков разработки и отладки пользовательских приложений с использованием модулей, их интеграция в систему.

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
-решать прикладные вопросы программирования и языка сценариев для создания программ;
- создавать и управлять проектом по разработке приложения;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Разработать дополнительный модуль для своего приложения.
2. Интегрировать модуль в систему, выполнить его отладку и тестирование.
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Краткие теоретические сведения:

Модуль - это часть программы, компилируемый отдельно от остальных её частей. Именно возможность раздельной компиляции и является основным преимуществом модулей. Простейшая модульность программы может достигаться за счёт применения процедур и функций, однако этого не всегда достаточно. Однако современные технологии программирования требуют возможности независимой разработки приложения коллективом разработчиков. Поэтому обычно приложение делится на множество файлов, так называемых «модулей». Несколько модулей, являющихся составными частями одной программы, объединяются в библиотеку.

Модульная система содержит рад преимуществ, среди которых можно выделить:

1. Удобная поддержка (Maintainability): По определению, хороший модуль является самодостаточным. Хорошо спроектированный модуль призван уменьшить зависимости частей вашей кодовой базы насколько это возможно, чтобы она могла расти и совершенствоваться не зависимо друг от друга. Обновить один модуль гораздо проще, когда он отделён от других частей кода.
2. Пространства имён (Namespacing): В JavaScript переменные которые находятся за пределами функций верхнего уровня считаются глобальными (каждый может получить к ним доступ). Поэтому очень распространено "Загрязнение пространства имён (namespace pollution)", где совершенно не связанный между собой код, связывают глобальные переменные. Совместное использование глобальных переменных в коде, который между собой не связан является существенным недостатком приложения. Модули позволяют избежать загрязнения глобального пространства имён, путём создания приватных пространств для переменных.
3. Повторное использование (Reusability): Возможность применять уже существующие модули в новых проектах.

Порядок выполнения работы:

1. Ознакомиться с теоретическим материалом.

2. Разработать для своего приложения модуль, реализующий новую функцию (самостоятельно продумать возможные варианты, дополняющие функционал приложения).
3. Интегрировать модуль в приложение, выполнить отладку, протестировать работу приложение на нескольких тестовых примерах. Оценить работоспособность модуля.
4. Составить краткий отчёт, содержащий описание процесса создания и отладки проекта.

Форма представления результата:

Выполненное задание, отчет о работе.

Критерии оценки:

Отлично - Выбранная для реализации в модуле функция дополняет функционал приложения, является полезной; правильно разработан алгоритм реализации модуля; модуль интегрирован с систему. выполнена отладка и тестирование приложения для различных вариантов входных параметров. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю.

Хорошо - Выбранная для реализации в модуле функция дополняет функционал приложения, является полезной; алгоритм для реализации модуля имеет ошибки; модуль интегрирован с систему. выполнена отладка и тестирование приложения для различных вариантов входных параметров. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю.

Удовлетворительно - Выбранная для реализации в модуле функция не является дополнением к уже существующему приложению; алгоритм для реализации модуля содержит существенные ошибки; модуль интегрирован с систему; выполнена отладка и тестирование приложения для различных вариантов входных параметров. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 14 Программирование обмена сообщениями между модулями

Цель: закрепление практических навыков разработки и отладки пользовательских приложений с использованием модульной структуры и организация взаимодействия между модулями.

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- использовать современное программное обеспечение;
- создавать и управлять проектом по разработке приложения;
- использовать языки структурного, объектно-ориентированного программирования и языка сценариев для создания независимых программ;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Разработать модуль для приложения «Калькулятор», реализующий статистические функции.
2. Выполнить отладку приложения с новым модулем, протестировать его работу.
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Краткие теоретические сведения:

Для того, чтобы модули «видели» код друг друга в JavaScript существует операция, называемая ***импорт***.

Например, существует модуль с именем "math.js":

```
const pi = 3.14;
const e = 2.718;

const square = (x) => {
  return x * x;
};

const surfaceArea = (r) => {
  return 4 * pi * square(r);
};
```

Для того, чтобы импортировать этот модуль в файл «primer.js»необходимо сделать следущее:

```
import { surfaceArea, square } from './math.js';

const surfaceOfMars = surfaceArea(3390);
const surfaceOfMercury = surfaceArea(2440);
const yearSquared = square(2017);
```

Ключевое слово **import**, затем список того, что мы хотим в фигурных скобках, а затем название модуля. Файл math.js находится в той же директории, что и текущий, поэтому "точка-слеш" говорит интерпретатору смотреть в текущей директории.

При этом модуль "math.js" должен ***экспортировать*** данные:

```
export const pi = 3.14;
export const e = 2.718;
```

```
export const square = (x) => {
  return x * x;
};

export const surfaceArea = (r) => {
  return 4 * pi * square(r);
};
```

В данном примере мы экспортируем всё.

Мы можем пометить любое объявление как экспортируемое, будь то переменная, функция или класс. разместив export перед ним.

Возвратимся к "primer.js". Допустим, мы хотим импортировать что-то еще. Мы можем просто добавить названия в список:

```
import { surfaceArea, square, pi, e } from './math.js';
```

Или импортировать всё сразу:

```
import * as mathematics from './math.js';
```

В этом случае мы импортируем весь модуль и даем ему название "mathematics". Теперь у нас есть доступ ко всему экспортированному из модуля math, но нам нужно сослаться на всё это через название "mathematics" таким способом:

```
import * as mathematics from './math.js';

const surfaceOfMars = mathematics.surfaceArea(3390);
const surfaceOfMercury = mathematics.surfaceArea(2440);
const yearSquared = mathematics.square(2017);
```

Часто требуется экспортировать из модуля что-то одно. Существет специальный механизм, который называется "экспорт по-умолчанию" и вы можете экспортировать с помощью него только что-то одно. Но экспортированную по умолчанию вещь проще импортировать.

```
const pi = 3.14;
const e = 2.718;

const square = (x) => {
  return x * x;
};

const surfaceArea = (r) => {
  return 4 * pi * square(r);
};
export default surfaceArea;
```

Просто напишите код, как обычно, без специально указанных экспортов, а в конце выполните "export default «имя экспортируемого элемента». В данном случае мы экспортируем функцию surfaceArea.

Импорт по-умолчанию выглядит так:

```
import surfaceArea from './math.js';

const surfaceOfMars = surfaceArea(3390);
```

Кроме описанных возможностей экспортирующие и импортирующие механизмы включают больше функциональности, вроде изменения названий в процессе импортирования, множество типов экспорта из единого модуля и другое.

Порядок выполнения работы:

1. Ознакомиться с теоретическим материалом.
2. Разработать модуль для приложения «Калькулятор», реализующий статистические функции.
3. Осуществить импорт и экспорт модуля, и его отдельных функций в приложение.
4. Выполнить отладку приложения с импортируемыми и экспортируемыми элементами, протестировать его работу.
5. Составить в электронном виде отчет о работе.
6. Сдать выполненную работу.

Форма представления результата:

Выполненное задание, отчет о работе.

Критерии оценки:

Отлично – Созданный модуль реализует работу статистических функций,; правильно разработан алгоритм импорта и экспорта модуля и его функций в приложение; выполнена отладка и тестирование приложения для различных вариантов. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю.

Хорошо - Созданный модуль реализует работу не всех статистических функций, в процессе импорта и экспорта модуля и его функций в приложение допущены незначительные ошибки; выполнена отладка и тестирование приложения для различных вариантов. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю.

Удовлетворительно - Созданный модуль реализует работу не всех статистических функций или функции работают неправильно, в процессе импорта и экспорта модуля и его функций в приложение допущены значительные ошибки; выполнена отладка и тестирование приложения для различных вариантов. Составлен краткий отчёт, содержащий описание процесса создания и отладки проекта. Работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 15 Организация файлового ввода-вывода данных

Цель: закрепление практических навыков разработки и отладки пользовательских приложений с организацией файлового ввода-вывода данных

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- определять задачи для поиска информации;
- определять необходимые источники информации;
- читать, понимать и находить необходимые технические данные и инструкции в руководствах в любом доступном формате;
- создавать и управлять проектом по разработке приложения;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Создать текстовый документ, содержащий справочную информацию об организации файлового ввода-вывода средствами JavaScript.
2. Создать HTML документ, представляющий найденную справочную информацию в виде таблиц.
3. Составить в электронном виде отчет о работе.
4. Сдать выполненную работу.

Краткие теоретические сведения:

JavaScript предоставляет класс File, который даёт приложению возможность записывать в файловой системе сервера. Это используется для генерации постоянных HTML-файлов и хранения информации без использования сервера БД. Одним из важнейших преимуществ хранения информации в файле вместо JavaScript - объектов является то, что информация сохраняется даже при отказе сервера.

Создание File-Объекта

Чтобы создать экземпляр класса File, используйте стандартный синтаксис JavaScript для создания объекта:

fileObjectName = new File("path");

Здесь fileObjectName это имя, по которому идет обращение к файлу, а path это полный путь к файлу. Этот путь должен быть указан в формате серверной файловой системы, а не URL.

Вы можете отобразить имя файла, используя функцию write с File -объектом в качестве аргумента. Например, следующий оператор выводит имя файла:

x = new File("\path\file.txt"); write(x);

Открытие и Закрытие Файла

После создания File -объекта Вы можете использовать метод open для открытия файла и чтения и записи. Метод open имеет следующий синтаксис:

result = fileObjectName.open("mode");

Это метод возвращает true, если операция прошла успешно, и false в ином случае. Если файл уже открыт, операция терпит неудачу, и оригинальный файл остаётся открытым.

Параметр mode это строка, специфицирующая режим открытия файла. В таблице описаны эти режимы.

Когда приложение заканчивает использование файла, оно может закрыть его, вызвав метод close. Если файл не открыт, close терпит неудачу.

Блокировка Файлов

Часто доступ ко многим приложениям могут выполнять одновременно многие пользователи.

Чтобы предотвратить модификацию файла одновременно несколькими пользователями, используйте один из механизмов блокирования, предоставляемых службой Session Management Service. Если один пользователь блокировал файл, другие пользователи приложения должны ждать, пока файл не будет разблокирован. В общем это означает, что lock должна предшествовать всем файловым операциям; после выполнения операций должно выполняться unlock (разблокирование).

Если только одно приложение может модифицировать данный файл, Вы можете получать блокировку в объекте project. Если более чем одно приложение может иметь доступ к одному и тому же файлу, получайте блокировку в объекте server.

Например, у Вас создан файл myFile. Затем Вы может использовать его так:

```
if ( project.lock() ) {
myFile.open("r");
// ... файл используется ...   myFile.close();

project.unlock();
}
```

Таким образом, только один пользователь приложения может изменять файл в данный момент времени. Для более тонкого управления блокировкой Вы можете создать Ваш собственный экземпляр класса Lock для управления доступом к данному файлу.

Класс File имеет несколько методов, которые можно использовать после открытия файла:

- Позиционирование: setPosition, getPosition, eof. Это методы для установки и получения текущей позиции указателя в файле и для определения, не находится ли указатель в конце файла.
- Чтение из файла: read, readln, readByte.
- Запись в файл: write, writeln, writeByte, flush.
- Конвертация двоичного и текстового форматов: byteToString, stringToByte. Конвертируют одно число в символ и наоборот.
- Информационные методы: getLength, exists, error, clearError. Для получения информации о файле и для получения и очистки error-статуса.

Эти методы описаны в последующих разделах.

Порядок выполнения работы:

1. Ознакомиться с теоретическим материалом.
2. Найти информацию о режимах доступа к файлам.
3. Найти информацию о работе методов класса File
4. Используя методы работы с файлами JavaScript создать HTML документ, представляющий найденную справочную информацию в виде таблиц.
5. Составить в электронном виде отчет о работе.
6. Сдать выполненную работу.

Форма представления результата:

Выполненное задание, текстовый файл, отчет о работе.

Критерии оценки:

Отлично - найдена вся необходимая справочная информация; сформирован HTML документ, представляющий информацию в требуемом виде; сделан отчет о выполненной работе работа сдана преподавателю.

Хорошо - найдена не вся необходимая справочная информация; сформирован HTML документ, представляющий информацию в требуемом виде; сделан отчет о выполненной работе работа сдана преподавателю.

Удовлетворительно – частично найдена необходимая справочная информация; сформирован HTML документ, представляющий информацию, но не организовано табличное представление; сделан отчет о выполненной работе работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 16 Разработка модулей экспертной системы

Цель: ознакомление с основами создания экспертных систем; закрепление практических навыков разработки и отладки пользовательских приложений

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- определять задачи для поиска информации;
- определять необходимые источники информации;
- читать, понимать и находить необходимые технические данные и инструкции в руководствах в любом доступном формате;
- создавать и управлять проектом по разработке приложения;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Разработать алгоритм и реализовать модуль для организации пользовательского взаимодействия с эс.
2. Составить в электронном виде отчет о работе.
3. Сдать выполненную работу.

Краткие теоретические сведения:

Основным назначением ЭС является разработка программных средств, которые при решении задач, трудных для человека, получают результаты, не уступающие по качеству и эффективности решения, решениям получаемым человеком-экспертом. ЭС используются для решения так называемых неформализованных задач, общим для которых является то, что:

- задачи не могут быть заданы в числовой форме;
- цели нельзя выразить в терминах точно определенной целевой функции;
- не существует алгоритмического решения задачи;
- если алгоритмическое решение есть, то его нельзя использовать из-за
- ограниченности ресурсов (время, память).

Кроме того, неформализованные задачи обладают ошибочностью, неполнотой, неоднозначностью и противоречивостью как исходных данных, так и знаний о решаемой задаче.

Экспертная система - это программное средство, использующее экспертные знания для обеспечения высокоэффективного решения неформализованных задач в узкой предметной области. Основу ЭС составляет база знаний (БЗ) о предметной области, которая накапливается в процессе построения и эксплуатации ЭС. Накопление и организация знаний - важнейшее свойство всех ЭС.

При разработке экспертных систем часто используется концепция быстрого прототипа. Суть её в следующем: поначалу создается не экспертная система, а её прототип, который обязан решать узкий круг задач и требовать на свою разработку незначительное время. Прототип должен продемонстрировать пригодность будущей экспертной системы для данной предметной области, проверить правильность кодировки фактов, связей и стратегий рассуждения эксперта. Он также дает возможность инженеру по знаниям привлечь эксперта к активной роли в разработке экспертной системы. Размер прототипа – несколько десятков правил.

На сегодняшний день сложилась определенная технология разработки экспертных систем, включающая 6 этапов:

1. Идентификация. Определяются задачи, которые подлежат решению. Планируется ход разработки прототипа экспертной системы, определяются: нужные ресурсы (время, люди, ЭВМ и т.д.), источники знаний (книги, дополнительные специалисты, методики), имеющиеся аналогичные экспертные системы, цели (распространение опыта, автоматизация рутинных действий и др.), классы решаемых задач и т.д. На этом же этапе разработки экспертных систем проходит извлечение знаний.

2. Концептуализация. Выявляется структура полученных знаний о предметной области. Определяются: терминология, перечень главных понятий и их атрибутов, структура входной и выходной информации, стратегия принятия решений и т.д. Концептуализация - это разработка неформального описания знаний о предметной области в виде графа, таблицы, диаграммы либо текста, которое отражает главные концепции и взаимосвязи между понятиями предметной области.

3. Формализация. На этапе формализации все ключевые понятия и отношения, выявленные на этапе концептуализации, выражаются на некотором формальном языке, предложенном (выбранном) инженером по знаниям. Здесь он определяет, подходят ли имеющиеся инструментальные средства для решения рассматриваемой проблемы или необходим выбор другого инструментария, или требуются оригинальные разработки.

4. Реализация. Создается прототип экспертной системы, включающий базу знаний и другие подсистемы. На данном этапе применяются следующие инструментальные средства: программирование на обычных языках (Паскаль, Си и др.), программирование на специализированных языках, применяемых в задачах искусственного интеллекта (LISP, FRL, SmallTalk и др.) и др. Четвертый этап разработки экспертных систем в какой-то степени является ключевым, так как здесь происходит создание программного комплекса, демонстрирующего жизнеспособность подхода в целом.

5. Тестирование. Прототип проверяется на удобство и адекватность интерфейсов ввода-вывода, эффективность стратегии управления, качество проверочных примеров, корректность базы знаний. Тестирование – это выявление ошибок в выбранном подходе, выявление ошибок в реализации прототипа, а также выработка рекомендаций по доводке системы до промышленного варианта.

6. Опытная эксплуатация. Проверяется пригодность экспертной системы для конечных пользователей. По результатам этого этапа может потребоваться существенная модификация экспертной системы.

Процесс разработки экспертной системы не сводится к строгой последовательности перечисленных выше этапов. В ходе работ приходится неоднократно возвращаться на более ранние этапы и пересматривать принятые там решения. В среднем на разработку каждого этапа уходит от 2 до 4 - х месяцев.

Архитектура экспертной системы.

1. *База знаний.* Составляет основу ЭС, хранит множество фактов и набор правил, полученных от экспертов, из специальной литературы. БЗ отличается от базы данных тем, что в базе данных единицы информации представляют собой не связанные друг с другом сведения, формулы, теоремы, аксиомы. В БЗ те же элементы уже связаны как между собой, так и с понятиями внешнего мира. Информация в БЗ - это все необходимое для понимания, формирования и решения проблемы. Она содержит два основных элемента: факты (данные) из предметной области и специальные эвристики или правила, которые управляют использованием фактов при решении проблемы. Знания могут быть представлены несколькими способами: логической моделью, продукциями, фреймами и семантическими сетями.

2. *Машина логического вывода* (МЛВ). Главным в ЭС является машина логического вывода, осуществляющая поиск в базе знаний для получения решения. Она манипулирует информацией из БЗ, определяя в каком порядке следует выявлять взаимосвязи и делать

выводы. МЛВ используются для моделирования рассуждений, обработки вопросов и подготовки ответов.

3. *Интерфейс пользователя*. ЭС содержат языковой процессор для общения между пользователем и компьютером. Это общение может быть организовано с помощью естественного языка, сопровождаться графикой или многооконным меню. Интерфейс пользователя должен обеспечивать два режима работы: режим приобретения знаний и режим решения задач. В режиме приобретения знаний эксперт общается с ЭС при посредничестве инженера знаний. В режиме решения задач ЭС для пользователя является или просто носителем информации (справочником), или позволяет получать результат и объясняет способ его получения. *Эксперты* поставляют знания в экспертную систему и оценивают правильность получаемых результатов.

В разработке эс участвуют:

- Инженер по знаниям - специалист по искусственному интеллекту, выступающий в роли промежуточного буфера между экспертом и базой знаний. Помогает эксперту выявить и структурировать знания.

- Программисты разрабатывают программное обеспечение экспертной системы и осуществляют его сопряжение со средой, в которой оно будет использоваться.

Пользователь - специалист предметной области, для которого предназначена система, обычно его квалификация недостаточно высока, и поэтому он нуждается в помощи и поддержке своей деятельности со стороны экспертной системы.

При разработке структуры эс необходимо обеспечить разделение методов создания и хранения информации о структуре и содержании предметной области от методов ее обработки.

В упрощённом виде структуру эс можно представить в виде набора модулей, реализующих различные задачи:

- Модуль управления базами знаний. Данный модуль реализует пользовательский интерфейс доступа к базам знаний и предназначен для работы (выполнение операций создания, модификации и удаления) с фактами и правилами, представленными в обобщенном виде. При этом правила и факты могут быть как абстрактными (образцы правил и фактов), так и конкретными (экземпляры правил и фактов). Факты и правила группируются согласно их предметной классификации, образуя базы знаний и формируя таким образом сферы своего применения. Каждая база знаний имеет уникальное имя и рассматривается компонентом в качестве предоставляемой им функции, доступ к которой осуществляется через унифицированный интерфейс компонента. Модуль разработан с использованием языков HTML, CSS, JavaScript и PHP.

- Одним из основных модулей компонента продукционной экспертной системы является продукционная машина вывода, осуществляющая процесс рассуждения (логического вывода) по правилам. Наиболее рациональным представляется реализация данного компонента на основе уже имеющейся машины вывода. В настоящее время существуют как коммерческие, так и свободно распространяемые оболочки продукционных экспертных систем (JESS [4], CLIPS [5], OPS5 [6]), с помощью которых можно реализовать механизм рассуждения на основе продукций.

- Модуль связи с web-сервисом. Реализует пользовательский интерфейс доступа к web-сервису экспертной системы. Передает машине вывода идентификатор базы знаний либо список правил и фактов в формате машины вывода.

- Модуль взаимодействия с БД – программный интерфейс доступа к БД. Реализует функции соединения с БД, отображения всех таблиц, доступа к их содержимому, а также выполнения стандартных операций и запросов (добавления, модификации, удаления, поиска).

Порядок выполнения работы:

1. Ознакомиться с теоретическим материалом.
2. На основе индивидуального задания разработать и создать модуль, позволяющий получить от сервера набор данных в некотором формате с описанием состава формы.
3. Реализовать эту форму, осуществить ввод данных и передать вводимые значения на сервер.
4. Составить в электронном виде отчет о работе.
5. Сдать выполненную работу.

Форма представления результата:

Выполненное задание, текстовый файл, отчет о работе.

Критерии оценки:

Отлично – модуль разработан и создан по заданию, форма соответствует описанию, ввод осуществлен, данные переданы; сделан отчет о выполненной работе работа сдана преподавателю.

Хорошо - модуль разработан и создан по заданию, с незначительными неточностями, в форме есть ошибки, ввод осуществлен, данные переданы; сделан отчет о выполненной работе работа сдана преподавателю.

Удовлетворительно - модуль разработан частично или не соответствует заданию, ошибки содержатся в реализации формы, ввода, передаче данных, сделан отчет о выполненной работе работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.

Лабораторная работа № 17 Создание сетевого сервера и сетевого клиента.

Цель: получение практического опыта создания сетевого сервера и сетевого клиента средствами Node.js

Выполнив работу, Вы будете:

уметь:

- работать с инструментальными средствами обработки информации;
- осуществлять выбор модели и средства построения информационной системы и программных средств;
- проектировать и разрабатывать систему по заданным требованиям и спецификациям;

Материальное обеспечение:

Персональный компьютер, Visual Studio Code

Задание:

1. Разработать алгоритм и реализовать программы создания сетевого сервера и сетевого клиента.
2. Составить в электронном виде отчет о работе.
3. Сдать выполненную работу.

Краткие теоретические сведения:

Node.js — это серверная платформа. Основная задача сервера — как можно быстрее и эффективнее обрабатывать запросы, поступающие от клиентов, в частности — от браузеров.

Методы

net.createServer ([options] [, connectionListener]) — Создает новый TCP-сервер. Аргумент connectionListener автоматически устанавливается как прослушиватель для события «connection».

net.connect(options[, connectionListener]) — Метод Фабрики, который возвращает новый «net.Socket» и подключается к указанному адресу и порту.

net.createConnection(options[, connectionListener]) — Метод Фабрики, который возвращает новый «net.Socket» и подключается к указанному адресу и порту.

net.connect(port[, host][, connectListener]) — Создает TCP-соединение с портом на хостинге. Если параметр host опущен, предполагается «localhost». Параметр connectListener будет добавлен в качестве прослушивателя для события «connect». Это метод Фабрики, который возвращает новый «net.Socket».

net.createConnection(port[, host][, connectListener]) — Создает TCP-соединение с портом на хостинге. Если параметр host опущен, предполагается «localhost». Параметр connectListener будет добавлен в качестве прослушивателя для события «connect». Это метод Фабрики, который возвращает новый «net.Socket».

net.connect(path[, connectListener]) — Создает подключение сокета Unix к пути. Параметр connectListener будет добавлен в качестве прослушивателя для события «connect». Это метод Фабрики, который возвращает новый «net.Socket».

net.createConnection (path [, connectListener]) — Создает подключение сокета Unix к пути. Параметр connectListener будет добавлен в качестве прослушивателя для события «connect». Это метод Фабрики, который возвращает новый «net.Socket».

net.isIP(input) — Проверяет, является ли input IP-адресом. Возвращает 0 для недопустимых строк, 4 — для адресов IP версии 4 и 6 — для адресов IP версии 6.

net.isIPv4(input) — Возвращает true, если input является IP-адресом версии 4, иначе возвращает false.

net.isIPv6(input) — Возвращает true, если input является IP-адресом версии 6, иначе возвращается false.

Класс — net.Server

Этот класс используется для создания TCP или локального сервера.

Методы

server.listen(port[, host][, backlog][, callback]) — Начинает принимать соединения на указанном порте и хосте. Если параметр host опущен, сервер будет принимать соединения, направленные на любой IPv4-адрес (INADDR_ANY). При значении порта равном нулю назначается случайный порт.

server.listen(handle[, callback]) — Запускает локальный сервер сокетов, который прослушивает соединения по данному пути.

server.listen(options[, callback]) — Объект handle может быть задан и как сервер, и как сокет (в обоих случаях с базовым элементом _handle) или как объект {fd: <n>}. Это указывает серверу принимать соединения по указанному дескриптору, но предполагается, что это дескриптор файла или handle уже привязан к порту или доменному сокету. Прослушивание дескриптора файла не поддерживается в Windows.

server.listen(options[, callback]) — Параметры порта, хоста и бэклогов, а также дополнительная функция обратного вызова ведут себя так же, как и при вызове server.listen(port, [host], [backlog], [callback]). Альтернативно, параметр пути может использоваться для указания сокета UNIX.

server.close([callback]) — Закрывается, когда все соединения завершаются, и сервер выдает событие ‘close’.

server.address() — Возвращает связанный адрес, имя семейства адресов и порт сервера, как их предоставляет операционная система.

server.unref() — Вызов unref на сервере позволяет программе выйти, если это единственный активный сервер в системе событий. Если сервер уже не очищен, то новый вызов unref не будет иметь никакого эффекта.

server.ref() — И в противоположность предыдущему методу, вызов ref на ранее очищенном сервере не позволит программе выйти, если это единственный активный сервер (поведение по умолчанию). Если сервер связан, то вызов новый ref не будет иметь никакого эффекта.

server.getConnections(callback) — Асинхронно получает количество параллельных подключений на сервере. Работает, когда сокеты отправляются на форки. Обратный вызов должен принимать два аргумента err и count.

События

listening — Запускается, когда сервер был установлен после вызова server.listen.

connection — Запускается при создании нового соединения. Объект socket, объект connection доступны для обработчика событий. Socket — это экземпляр net.Socket.

close — Запускается при закрытии сервера. Обратите внимание: если соединения существуют, это событие не будет выдаваться до тех пор, пока все соединения не будут завершены.

error — Запускается при возникновении ошибки. Событие ‘close’ будет вызываться непосредственно после этого события.

Класс — net.Socket

Этот объект является абстрактным представлением TCP или локального сокета. net.Socket реализует дуплексный интерфейс Stream. Они могут быть созданы пользователем и использоваться как клиент (с помощью connect()), или они могут быть созданы Node и переданы пользователю через событие ‘connection’ сервера.

События

net.Socket является эмитером событий и запускает следующие события.

lookup — Запускается после обработки имени хоста, но до соединения. Не применимо к сокетам UNIX.

connect — Запускается, когда соединение сокета успешно установлено.

data — Запускается при получении данных. Данные аргумента представляют собой буфер или строку. Кодировка данных устанавливается с помощью socket.setEncoding().

end — Выдается, когда другой конец сокета отправляет пакет FIN.

timeout — Запускается, если сокет простаивает из-за неактивности. Используется только для уведомления о простое сокета. Пользователь должен вручную закрыть соединение.

drain — Запускается, когда буфер записи очищается. Может использоваться для дроссельной загрузки.

error — Запускается при возникновении ошибки. Событие ‘close’ будет вызываться непосредственно после этого события.

close — Запускается после полного закрытия сокета. Аргумент has_error является логическим, указывающим, был ли сокет закрыт из-за ошибки передачи.

Свойства

net.Socket предоставляет множество важных свойств для лучше контроля взаимодействия с сокетами.

socket.bufferSize — Это свойство отображает количество символов, которые в настоящий момент буферизуются для записи.

socket.remoteAddress — Строковое представление удаленного IP-адреса. Например, «74 .125.127.100» или «2001: 4860: a005 :: 68».

socket.remoteFamily — Строковое представление удаленного семейства IP-адресов. «IPv4» или «IPv6».

socket.remotePort — Числовое представление удаленного порта. Например, 80 или 21.

socket.localAddress — Строковое представление локального IP-адреса, к которому подключается удаленный клиент. Например, если вы прослушиваете «0.0.0.0», а клиент подключается к «192.168.1.1», значение будет «192.168.1.1».

socket.localPort — Числовое представление локального порта. Например, 80 или 21.

socket.bytesRead — Количество принятых байтов.

socket.bytesWritten — Количество отправленных байтов.

Методы

new net.Socket([options]) — Создает новый объект сокета.

socket.connect(port[, host][, connectListener]) — Открывает соединение для данного сокета. Если заданы port и host, то сокет будет открыт как сокет TCP, если host опущен, предполагается localhost. Если задан путь, сокет будет открыт как сокет Unix для этого пути.

socket.connect(path[, connectListener]) — Открывает соединение для данного сокета. Если заданы port и host, то сокет будет открыт как сокет TCP, если host опущен, предполагается localhost. Если задан путь, сокет будет открыт как сокет Unix для этого пути.

socket.setEncoding([encoding]) — Установливает кодировку сокета как для считываемого потока.

socket.write(data[, encoding][, callback]) — Отправляет данные в сокет. Второй параметр указывает кодировку в случае, если это строчные данные — по умолчанию используется кодировка UTF8.

socket.end([data][, encoding]) — Наполовину закрывает сокет, т. е. отправляет пакет FIN. Возможно, что сервер будет продолжать отправлять некоторые данные.

socket.destroy() — Обеспечивает, чтобы в этом сокете не содержалось операций ввода-вывода данных. Необходимо только в случае ошибок (ошибка парсинга или подобная).

socket.pause() — Приостанавливает чтение данных. То есть, события «data» не будут запускаться. Используется для дросселирования загрузки.

socket.resume() — Возобновляет чтение после вызова pause().

socket.setTimeout (timeout [, callback]) — Устанавливает для сокета режим Timeout после timeout миллисекунд простоя сокета. По умолчанию net.Socket не устанавливается на Timeout.

socket.setNoDelay ([NODELAY]) — Отключает алгоритм Nagle. По умолчанию TCP-соединения используют алгоритм Nagle, они концентрируют данные перед отправкой. Значение true для noDelay немедленно отключает данные каждый раз, когда вызывается

socket.write(). По умолчанию для noDelay задано true.

socket.setKeepAlive ([enable] [, initialDelay]) — Включает/отключает функцию keep-alive и опционально установливает задержку до того момента, как к простаивающему сокету будет отправлен первый запрос keepalive. Для параметра enable по умолчанию задано значение false.

socket.address () — Возвращает связанный адрес, имя семейства адресов и порт сокета, как они предоставляются операционной системой. Возвращает объект с тремя свойствами, например { port: 12346, family: ‘IPv4′, address: ’127.0.0.1′ }.

socket.unref() — Вызов unref в сокете позволит программе выйти, если это единственный активный сокет в системе событий. Если сокет уже очищен, тогда новый вызов unref не окажет никакого влияния.

socket.ref() — В противоположность unref, вызов ref не позволит программе завершиться, если сокет остается единственным активным сокетом (поведение по умолчанию). Если сокет уже связан, то новый вызов ref не окажет никакого влияния.

Порядок выполнения работы:

1. Ознакомиться с теоретическим материалом.
2. Разработать алгоритм и реализовать модуль создания сетевого сервера.
3. Разработать алгоритм и реализовать модуль создания сетевого клиента.
4. Осуществить взаимодействие между сервером и клиентом на примере передачи данных для экспертной системы.
5. Составить в электронном виде отчет о работе.
6. Сдать выполненную работу.

Форма представления результата:

Выполненное задание, текстовый файл, отчет о работе.

Критерии оценки:

Отлично - разработаны модули серверной и клиентской части, осуществлена передача данных; сделан отчет о выполненной работе; работа сдана преподавателю.

Хорошо - в процессе разработки модулей серверной и клиентской части, передачи данных были ошибки; сделан отчет о выполненной работе; работа сдана преподавателю.

Удовлетворительно - модули серверной и клиентской части содержат значительные ошибки, нет передачи данных ; сделан отчет о выполненной работе; работа сдана преподавателю.

Неудовлетворительно - работа не выполнена.