МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Омский государственный технический университет»

ТЕОРИЯ И ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ

Учебное текстовое электронное издание
локального распространения

Омск
Издательство ОмГТУ
2014

Сведения об издании: 1, 2

Теория и технология программирования : метод. указания / [сост. А. Г. Белик, В. Н. Цыганенко] ; Минобрнауки России, ОмГТУ. – Омск : Изд-во ОмГТУ, 2014.

Приведены цели, задачи и требования к выполнению расчетно-графической работы по дисциплине «Теория и технология программирования». Описываются основные правила и методы анализа потоков данных, структурирования программ, алгоритмизации, кодирования и тестирования при разработке программного продукта. Излагаются основные этапы проектирования программ.

Предназначены для студентов направлений подготовки бакалавров 230100 «Информатика и вычислительная техника», 220100 «Системный анализ и управление», 231000 «Программная инженерия».

Рекомендовано редакционно-издательским советом
Омского государственного технического университета

1 электронный оптический диск

Оригинал-макет издания выполнен в Microsoft Office Word 2007
с использованием возможностей Adobe Acrobat X.

Минимальные системные требования:

- процессор Intel Pentium 1,3 ГГц и выше;
- оперативная память 256 Мб;
- свободное место на жестком диске 260 Мб;
- операционная система Microsoft Windows XP/Vista/7;
- разрешение экрана 1024×576 и выше;
- акустическая система не требуется;
- дополнительные программные средства Adobe Acrobat Reader 5.0 и выше.

Редактор *О. В. Маер*
Компьютерная верстка *А. Н. Кошелапова*

Сводный темплан 2014 г.
Подписано к использованию 25.03.14.
Объем 614 Кб.

Издательство ОмГТУ.
644050, г. Омск, пр. Мира, 11; т. 23-02-12
Эл. почта: info@omgtu.ru

1. МЕТОДИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТА

1.1. Цели и задачи дисциплины

В ходе иосвония курса «Теория и технология программирования» студенты должны изучить теоретические подходы, методологические основы и технологический процесс разработки программных продуктов, в том числе:

– процесс и технологический цикл производства программных продуктов;

– основные подходы к разработке программных продуктов: процедурное, логическое, функциональное, объектно ориентированное программирование;

– методы, технологии и инструментальные средства разработки программных продуктов, тестирования и отладки программ и программных модулей;

– документирование программных систем и виды программных документов;

– методологии структурного анализа и моделирования программных систем;

– способы организации коллективной работы по созданию программ;

– методы автоматизации проектирования программного обеспечения и инструментальные CASE-средства.

Изучение дисциплины предполагает самостоятельную разработку студентами программного продукта с освоением всех стадий и этапов проектирования. Самостоятельная работа студентов складывается из изучения теоретического материала по конспекту лекций и литературным источникам и выполнения расчетно-графической работы (РГР), направленной на освоение основных этапов проектирования прикладных программных систем. Общая постановка задачи на РГР может быть сформулирована как ***разработка технического проекта автоматизированной системы обработки информации и управления*** по индивидуальному заданию.

Техническое проектирование – стадия разработки конструкторской документации на изделие или стадия создания автоматизированной системы. Под техническим проектом понимается совокупность технических документов, которые содержат окончательные проектные решения по изделию (системе). Разработка технического проекта на автоматизированные системы осуществляется в соответствии с комплексом стандартов на автоматизированные системы (ГОСТ 34-й серии).

Техническое проектирование относится к наименее формализованным этапам создания программных систем. Это связано с тем, что разработчик должен выполнять большой объем работ исследовательского характера по изучению автоматизируемой предметной области, определению основных показателей эффективности, анализу существующих аналогов и прототипов. В результате возникает достаточно большое количество вариантов принимаемых проектных решений, из которых необходимо выбрать оптимальное.

Техническое проектирование предполагает широкое использование CASE-средств проектирования программных систем, которые применяются для автоматизации труда разработчиков. В первую очередь, это инструментальные средства анализа и проектирования, поддерживающие наиболее распространенные методологии проектирования, позволяющие формировать спецификации архитектуры и интерфейсов системы, алгоритмов и структуры данных [7].

1.2. Тематика и выполнение расчетно-графической работы

Расчетно-графическая работа (РГР) выполняется по индивидуальному заданию, тема и функциональное назначение программы согласовываются с ведущим преподавателем. Студент имеет право предложить собственную тему РГР, если им выполняются учебные исследовательские и проектные работы по различным дисциплинам, требующие разработки

технического проекта программного обеспечения. Темы РГР выбираются таким образом, чтобы при выполнении работы студенты могли приобрести практические навыки проектирования программных систем среднего уровня сложности. Рекомендуется ориентировать тематику РГР на разработку систем автоматизации таких видов деятельности, как учет и контроль состояния различных объектов, планирование и оперативное управление производственными и сервисными работами, анализ и поддержка принятия управленческих решений. В связи с этим желательно обеспечивать наличие у разрабатываемого продукта развитого пользовательского интерфейса. Объем РГР должен составлять 20–25 машинописных страниц.

Примерные темы РГР:

– «Система учета и контроля успеваемости студентов»;

– «Автоматизированная система складского учета»;

– «Автоматизированная система ремонта и сервисного обслуживания»;

– «Система автоматизации предприятия общественного питания».

В процессе выполнения РГР студенты должны:

1) разработать развернутое техническое задание на программный продукт;

2) разработать модель программной архитектуры системы;

3) провести анализ структуры данных для реализации предметной области программного продукта и разработать структуру информационных баз системы;

4) разработать основные алгоритмы обработки информации и управления;

5) разработать пользовательский интерфейс программной системы.

Техническое задание должно содержать основные требования к разработке. Оно разрабатывается в первую очередь – еще до начала выполнения основных работ по техническому проектированию программной системы. Оно должно быть согласовано с ведущим преподавателем и утверждено.

РГР оформляется в виде пояснительной записки и защищается перед ведущим преподавателем или комиссией. На защиту студент предоставляет техническое задание и пояснительную записку в отпечатанном виде, содержащую описание технического проекта и соответствующие иллюстрации. На защите студент коротко (3–5 мин) докладывает об основных проектных решениях, принятых в процессе разработки, и отвечает на вопросы.

1.3. Сроки выполнения отдельных видов работ по этапам

Выполнение РГР осуществляется в несколько этапов на протяжении учебного семестра.

Примерные сроки выполнения этапов РГР и представляемые преподавателю результаты приведены в таблице.

Этапы выполнения РГР

Этап		Сроки выполнения (№ нед)	Объем выполнения работы, %	Представляемые результаты
№	Содержание			
1	Формулировка темы проекта, разработка технического задания	2–3	10	Готовое к утверждению техническое задание
2	Анализ потоков данных, разработка модели информационной структуры данных	4–6	30	Диаграммы потоков данных, ER-диаграммы, описания структур данных
3	Функциональное моделирование и разработка программной архитектуры	7–9	50	Функциональные диаграммы, структурные карты программных компонентов
4	Разработка алгоритмов	10–12	70	Схемы алгоритмов

Окончание таблицы

5	Разработка пользовательского интерфейса	13–14	90	Граф состояний интерфейса и описание основных экранных форм
6	Подготовка отчета и защита	15–16	100	Полностью оформленный отчет в печатном варианте

В конце каждого этапа студент демонстрирует преподавателю результаты выполнения работы в виде диаграмм и фрагментов отчета, текстовых или графических материалов, иллюстрирующих разработку. Сначала разрабатывается техническое задание, которое утверждается ведущим преподавателем и является основанием для выполнения последующих этапов проектирования.

2. РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ

2.1. Соглашение между заказчиком и исполнителем

Спецификация (техническое задание) на программный продукт является результатом системного анализа и как самостоятельный документ имеет очень важное значение. Этот документ является формальным соглашением между заказчиком продукта и его разработчиками. Заказчиком программного продукта, выполняемого в рамках курсового проектирования, формально можно считать ведущего преподавателя.

Техническое задание (ТЗ) является основанием для принятия проектных решений и содержит основные требования к программной системе. Поэтому начинать проектирование можно только после утверждения его заказчиком. Основным критерием проверки и приемки РГР является степень соответствия формальных требований, изложенных в ТЗ, проектным решениям, представленным в техническом проекте.

В случае каких-либо конфликтных ситуаций, связанных с различным пониманием заказчиком и исполнителями степени «правильности» и «функциональности» созданного ПО, окончательное решение всегда принимается в строгом соответствии с ТЗ. Если какое-либо свойство продукта не было детально оговорено заказчиком в ТЗ, то претензии на отсутствие или недостаточную реализацию этого свойства будут, скорее всего, отвергнуты.

2.2. Рекомендуемая структура технического задания

Техническое задание к РГР должно быть составлено в соответствии с ЕСПД (ГОСТ 19.201–78) и включать следующие разделы (количество разделов относительно ГОСТ сокращено):

1. Введение.

Во введении необходимо кратко обосновать актуальность разработки. В нем также указывается наименование продукта и дается краткая характеристика области применения программной системы**.**

2. Назначение разработки.

В данном разделе объясняется, для чего предназначена данная разработка (более подробно, чем во введении), описывается функциональное и эксплуатационное назначение программного продукта, а также оговаривается круг лиц (специалистов), для автоматизации деятельности которых будет использоваться программный продукт.

3. Требования к программе.

Требования к программному продукту подразделяются на группы и указываются в соответствующих разделах.

3.1. *Требования к функциональным характеристикам* по составу выполняемых программной системой функций, характеристикам и форме представления входных и выходных данных.

3.2. *Требования к надежности*, такие как контроль входной и выходной информации, создание резервных копий промежуточных результатов и т. п.

3.3. *Требования к составу и параметрам технических средств* (необходимые параметры используемых ЭВМ – тип микропроцессора, объем памяти, наличие внешних устройств, например мыши).

3.4. *Требования к информационной и программной совместимости,* здесь при необходимости задаются методы решения, используемые языки программирования, а также операционная система и другие системные и пользовательские программные средства.

4. Требования к программной документации.

В этом разделе должен быть определен предварительный состав программной документации и, при необходимости, специальные требования к ней. Обязательными документами, подлежащими разработке в ходе проектирования программной системы, являются следующие: «Руководство пользователя», «Руководство системного администратора».

5. Стадии и этапы разработки.

Устанавливают необходимые стадии разработки, этапы и содержание работ, а также, как правило, сроки разработки. Применительно к изу-

чаемой дисциплине, данный раздел должен выполняться согласно срокам, изложенным в таблице (п. 1.3), с указанием конкретных дат.

6. Приложения.

В приложениях к ТЗ при необходимости приводят:

– перечень научно-исследовательских и других работ, обосновывающих разработку;

– схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые могут быть использованы при разработке.

Техническое задание на создание игровой программной системы в одном из разделов обязательно должно содержать правила игры, обычно они описываются в разделе «Назначение системы».

Техническое задание на создание справочной или информационно-поисковой системы должно содержать подробное описание исходных данных и результатов в подразделе 3.1. «Требования к функциональным характеристикам».

Техническое задание утверждается ведущим преподавателем. Образец технического задания можно посмотреть в [6].

3. АНАЛИЗ ПОТОКОВ И СТРУКТУР ДАННЫХ

3.1. Моделирование потоков данных

Диаграммы потоков данных позволяют специфицировать как функции создаваемого ПО, так и обрабатываемые им данные. При использовании этой модели систему представляют в виде иерархии диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. На каждом следующем уровне происходит уточнение процессов, пока очередной процесс не будет признан элементарным.

В основе модели лежат понятия внешней сущности, процесса, хранилища (накопителя) данных, потока данных [5, 7].

Для изображения диаграмм потоков данных чаще всего обращаются к нотации Гейна – Сарсона [9]. Данная нотация применяется в CASE-средстве для проведения структурного анализа систем *All Fusion Business Process Modeling (BPwin)*, которое рекомендуется к использованию при выполнении работ по анализу потоков данных проектируемой программной системы.

Над линией, изображающей поток данных, направление которого обозначают стрелкой, указывают, какая конкретно информация в данном случае передается (рис. 1).

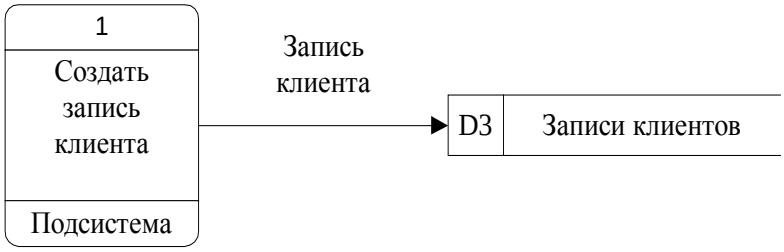

Рис. 1. Пример потока данных

Модель потоков данных представляет собой иерархическую структуру диаграмм, каждая из которых описывает систему, подсистему или процесс. Основным способом, применяемым для создания такой структу-

ры, является метод структурной декомпозиции, в ходе которой программная система разбивается на подсистемы, состоящие в свою очередь из заданной совокупности процессов (или других подсистем). Количество уровней в такой иерархии может быть любым. Процесс считается элементом модели, который является достаточно простым для разработки алгоритма его поведения и дальнейшей декомпозиции не подвергается.

Формальное решение о завершении детализации процесса принимается в следующих случаях:

– процесс взаимодействует с 2–3 потоками данных;

– возможно описание процесса последовательным алгоритмом;

– процесс выполняет единственную логическую функцию преобразования входной информации в выходную.

Построение иерархии диаграмм потоков данных начинают с диаграммы особого вида – *контекстной диаграммы*, которая определяет наиболее общий вид системы. На такой диаграмме показывают, как разрабатываемая система будет взаимодействовать с приемниками и источниками информации без указания исполнителей, т. е. описывают интерфейс системы с внешним миром. Обычно начальная контекстная диаграмма имеет форму звезды.

Если проектируемая система содержит большое количество внешних сущностей (более 10), имеет распределенную природу или включает уже существующие подсистемы, то строят *иерархии* контекстных диаграмм.

При разработке контекстных диаграмм происходит детализация функциональной структуры будущей системы, что особенно важно, если разработка ведется несколькими коллективами разработчиков.

Полученную таким образом модель системы проверяют на полноту исходных данных об объектах системы и изолированность объектов (отсутствие информационных связей с другими объектами). На следующем этапе каждую подсистему контекстной диаграммы детализируют при помощи диаграмм потоков данных.

В процессе детализации соблюдают правило балансировки – *при детализации подсистемы могут использоваться компоненты только тех подсистем, с которыми у разрабатываемой подсистемы существует информационная связь* (т. е. с которыми она связана потоками данных).

Для облегчения восприятия процессы детализируемой подсистемы нумеруют, соблюдая иерархию номеров: процессы, полученные при детализации процесса или подсистемы «1», должны нумероваться «1.1», «1.2» и т. д. Кроме этого, желательно размещать на каждой диаграмме от 3 до 6 процессов и не загромождать диаграммы деталями, не существенными на данном уровне.

Декомпозицию потоков данных необходимо осуществлять параллельно с декомпозицией процессов. Полная спецификация процессов включает описание *структур данных*, используемых как при передаче информации в потоке, так и при хранении в накопителе. Описываемые структуры данных могут содержать альтернативы, условные вхождения и итерации. Условное вхождение означает, что соответствующие элементы данных в структуре могут отсутствовать; альтернатива –в структуру может входить один из перечисленных элементов; итерация – элемент может повторяться некоторое количество раз.

Кроме того, для данных должен быть указан их тип. Для непрерывных данных могут указываться единицы измерений, диапазон значений, точность представления и форма физического кодирования, для дискретных данных – таблица допустимых значений.

Полученную законченную модель необходимо проверить на полноту и согласованность. Под согласованностью модели в данном случае понимают выполнение для всех потоков данных правила *сохранения информации*: все поступающие куда-либо данные должны быть считаны и записаны.

Пример. Разработать иерархию диаграмм потоков данных системы учета успеваемости студентов.

В качестве внешних сущностей для системы выступают декан, заместитель декана по курсу и сотрудник деканата. Определяем потоки данных между этими сущностями и системой.

Декан должен получать:

– сводку успеваемости по факультету (процент успеваемости групп, курсов и в целом по факультету) на текущий или указанный момент времени;

– полные сведения об учебе конкретного студента (успеваемость по всем изученным предметам всех завершенных семестров обучения с учетом пересдач).

Заместитель декана по курсу должен получать:

– сводку успеваемости по курсу (процент успеваемости по группам) на текущий или указанный момент;

– сведения о сдаче экзаменов и зачетов указанной группой;

– текущие сведения об успеваемости конкретного студента;

– полные сведения об учебе конкретного студента (успеваемость по всем изученным предметам всех завершенных семестров обучения с учетом пересдач);

– список задолжников по факультету, с указанием группы обучаемых и несданных ими предметов.

Сотрудник деканата должен обеспечивать:

– ввод списков студентов, зачисленных на первый курс;

– корректировку списков студентов в соответствии с приказами о зачислении, отчислении, переводе и т. п.;

– ввод учебных планов кафедр;

– ввод расписания сессии;

– ввод результатов сдачи зачетов и экзаменов на основании ведомостей и направлений.

Кроме того, сотрудник деканата должен иметь возможность получать:

– справку о прослушанных студентом предметах с указанием часов и итоговых оценок;

– приложение к диплому выпускника с указанием часов и итоговых оценок.

В результате получаем контекстную диаграмму, которая изображена на рис. 2 с применением нотации Гейна – Сарсона.

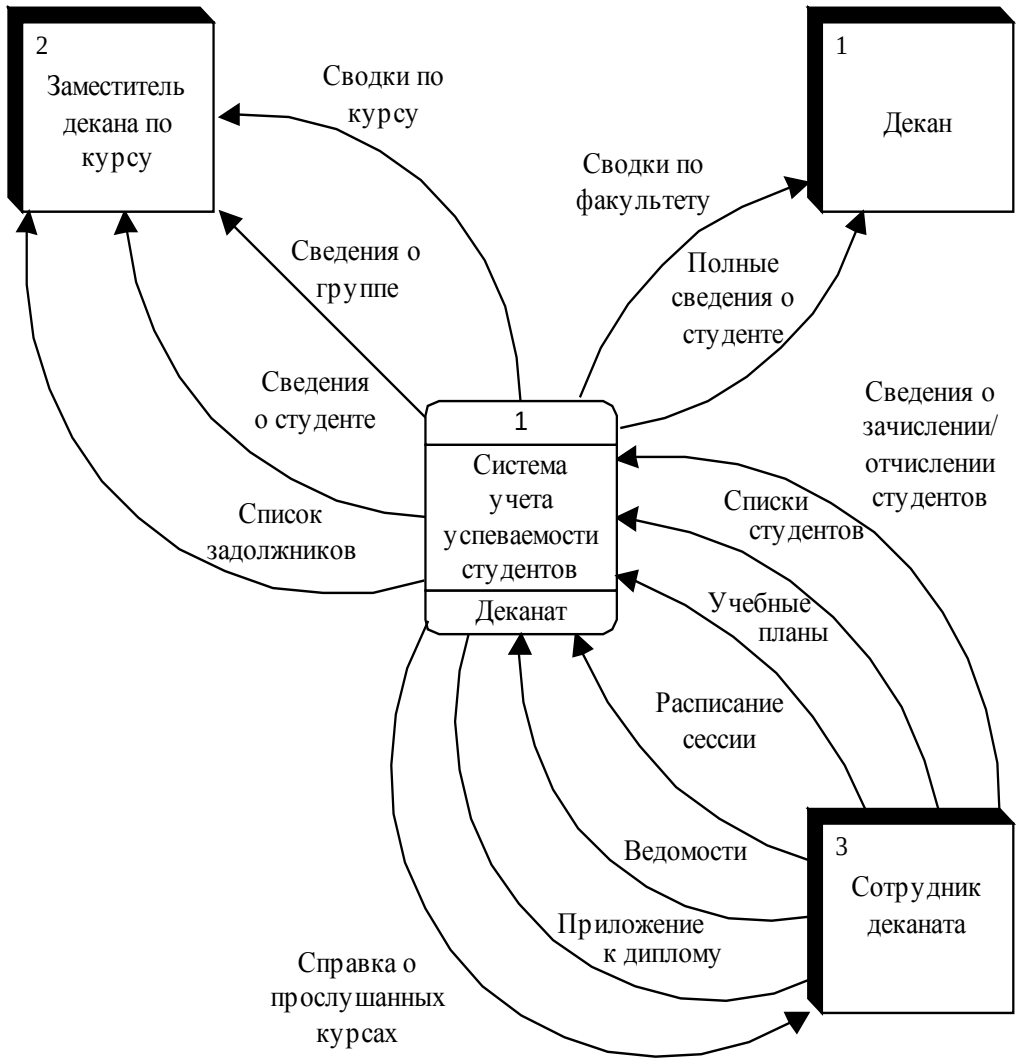

Рис. 2. Контекстная диаграмма системы учета успеваемости студентов

На рис. 3 представлена детализирующая диаграмма потоков данных, на которой выделены две подсистемы: Подсистема наполнения базы и Подсистема формирования отчетов, а также хранилище данных, которое может быть реализовано как с помощью средств СУБД, так и без них.

Дальнейшая детализация процессов осуществляется разработкой диаграмм потоков данных подсистем, включающих процессы, при помощи которых производятся операции проверки данных, составления запросов, формирования разных видов отчетов, аутентификации и т. д.

Рис. 3. Детализирующая диаграмма потоков данных второго уровня

На диаграмме потоков данных (ДПД) потоки данных представляются, как правило, в виде логических объектов, соответствующих документам или сообщениям конкретных форм и являющимися элементами информационного взаимодействия участников автоматизируемой деятельности. Полная спецификация потоков данных должна включать детальное описание структур данных, особенно при моделировании и проектировании баз данных.

3.2. Методы представления структур данных

Структурой данных называют совокупность правил и ограничений, которые отражают связи, существующие между отдельными частями (элементами) данных. Различают *абстрактные структуры* данных, используемые для уточнения связей между элементами, и *конкретные структуры*, применяемые для представления данных в программах. Диаграммы отношений компонентов данных в отличие от функциональных диаграмм предназначены для определения спецификаций структур данных программы.

Абстрактные структуры можно разделить на три группы: структуры, элементы которых не связаны между собой (множества, кортежи); структуры с неявными связями элементов (таблицы, структуры); структуры, связь элементов которых указывается явно (графы). Существенно, что в реальности возможно вложение структур данных, в том числе и разных типов, поэтому для их описания могут потребоваться специальные модели. В зависимости от описываемых типов отношений модели структур данных принято делить на иерархические и сетевые.

Сетевые модели основаны на графах, а потому позволяют описывать связность взаимодействующих компонентов независимо от вида отношения, в том числе комбинации множеств, таблиц и графов. Примером сетевой модели является модель «сущность-связь» (ER – Entity-Relationship), обычно используемая при разработке баз данных. Цель ER-моделирования данных состоит в обеспечении разработчика ИС концептуальной схемой базы данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую систему баз данных. При разработке такой модели данных определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи). Модели «сущность-связь» часто используются при проектировании реляционных баз данных [5, 7].

Иерархические модели позволяют описывать упорядоченные или неупорядоченные отношения *вхождения* элементов данных в компонент более высокого уровня, т. е. файла, множества, таблицы и их комбинации. Такие описания данных необходимы при проектировании внутренних программных переменных сложных типов, создании структуры записей для файлового хранения информации. К иерархическим моделям относят модель Джексона, для графического представления которой могут использоваться диаграммы Джексона, предложенные в составе методики проектирования программного обеспечения.

3.3. Диаграмма Джексона

В основе диаграмм Джексона лежит предположение о том, что структуры данных, так же как и программ, можно строить из элементов с использованием всего трех основных конструкций, определяющих типы связей: последовательности, выбора и повторения. Нотации конструкций различаются специальными символами в правом верхнем углу блоков элементов. В изображении последовательности символ отсутствует. В изображении выбора ставится символ «о» – сокращение английского or (или). Конструкции последовательности и выбора должны содержать по два или более элемента второго уровня. В изображении повторения в блоке повторяющегося элемента ставится символ «*».

Так схема, показанная на рис. 4, *а*, означает, что конструкция A состоит из элементов B, C и D, следующих в указанном порядке. Схема на рис. 4, *б* означает, что конструкция S состоит либо из элемента P, либо из элемента Q, либо из элемента R. Схема, изображенная на рис. 4, *в*, показывает, что конструкция I состоит из 0 или более элементов X.

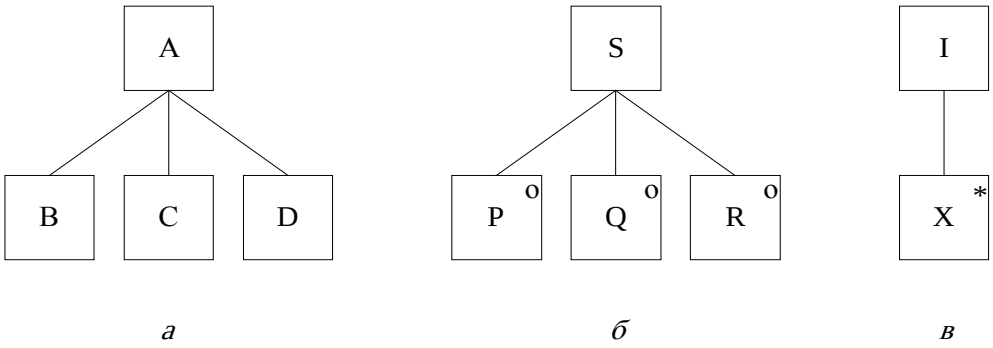

Рис. 4. Нотация Джексона для представления конструкций:
а – последовательность; *б* – выбор; *в* – повторение

В том случае, если необходимо показать, что конструкция повторения должна включать один или более элементов, используют комбинацию из двух структур последовательности и повторения (рис. 5).

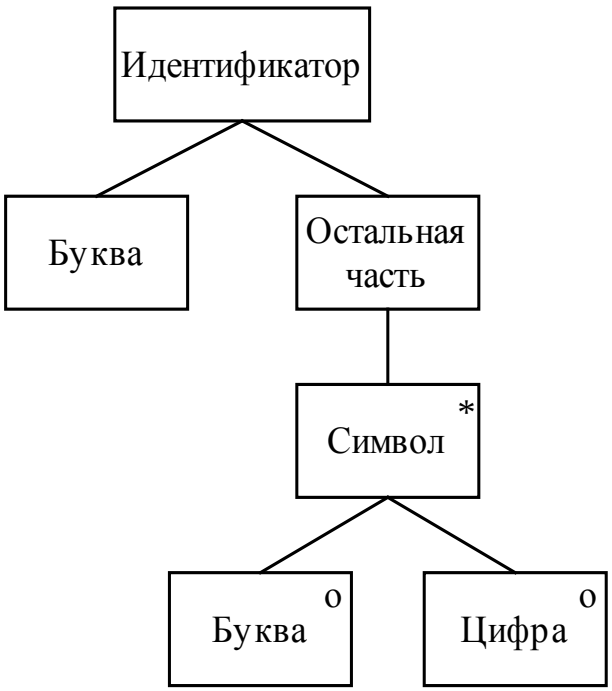

Рис. 5. Пример описания конструкции, в которой встречается повторение

Пример. Рассмотрим описание структуры данных файла «Электронная ведомость», содержащего сведения о сдаче экзаменов студентами. Файл состоит из записей о результатах сдачи сессии студентами одной группы. Он имеет следующую структуру: номер группы, записи об успеваемости студентов и специальный символ «конец файла». На рис. 6 показано, как выглядит описание данной структуры с использованием диаграммы Джексона.

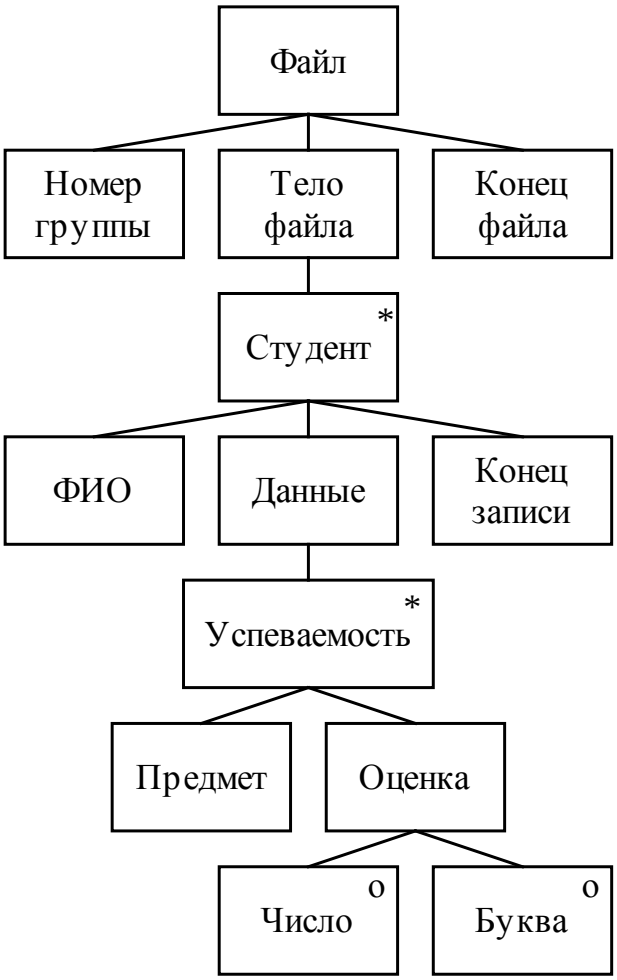

Рис. 6. Описание структуры файла в виде диаграммы Джексона

Таким образом, каждая конструкция представляется в виде двухуровневой иерархии, на верхнем уровне которой расположен блок конструкции, а на нижнем – блоки элементов.

4. РАЗРАБОТКА ПРОГРАММНОЙ АРХИТЕКТУРЫ

4.1. Структурная декомпозиция программной системы

В данном разделе проводится анализ предметной области задачи и ее разбиение (декомпозиция) в соответствии с выбранной технологией, т. е. создается структурная схема системной архитектуры будущего продукта и описывается взаимодействие ее функциональных элементов [1].

Для решения этой задачи могут быть использованы различные методологии:

– методология функционального моделирования в структурном анализе SADT (Structured Analysis and Design Technique) [2, 5];

– моделирование потоков данных [2, 4];

– логическое и физическое моделирование структуры программной системы с использованием методологии UML [2, 3];

– структурные карты Константайна [2].

Основным средством представления программной архитектуры системы является *структурная схема* – схема, отражающая состав и взаимодействие по управлению частей разрабатываемого продукта. При объектной декомпозиции такими частями являются объекты (классы), при структурной декомпозиции – подпрограммы (процедуры или функции).

Схема процедурной декомпозиции системы на примере программы построения графиков функции приведена на рис. 7 .

Обычно при выборе элементов структуры программного комплекса учитываются следующие параметры: объем и типы данных, а также основные операции над данными (хранение, поиск, сортировка) и частота обращения к ним в процессе выполнения программы. Если возможны варианты, то производится их оценка по объему требуемой памяти и вычислительной сложности выполнения основных операций.

Структурный подход предлагает осуществлять декомпозицию программ методом пошаговой детализации. Результат декомпозиции – структурная схема программы – представляет собой многоуровневую иерархическую схему взаимодействия подпрограмм по управлению. Минимально

такая схема отображает два уровня иерархии, т. е. показывает общую структуру программы. Однако тот же метод позволяет получить структурные схемы с большим количеством уровней.

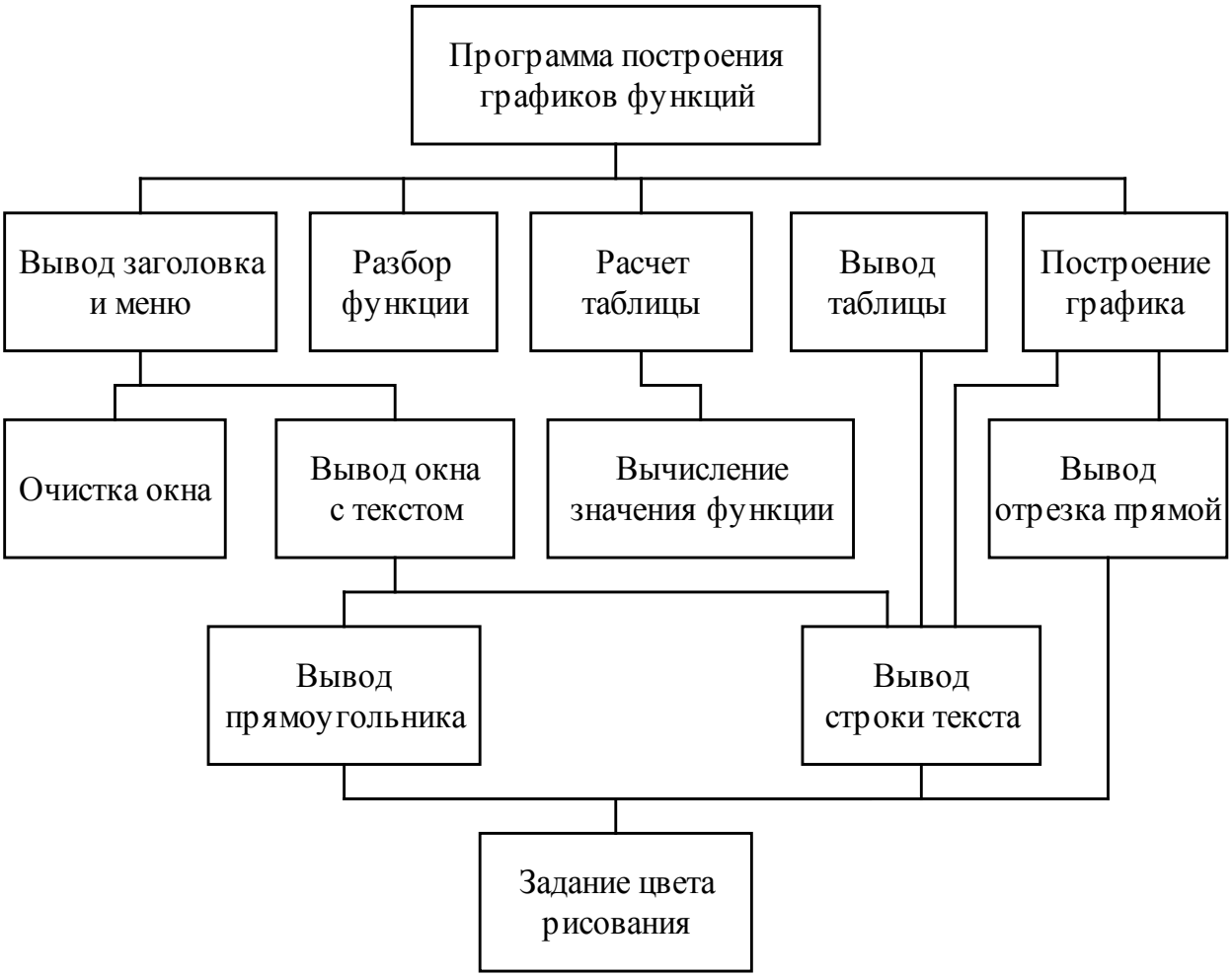

Рис. 7. Структурная схема программного продукта

Метод пошаговой детализации реализует нисходящий подход и базируется на основных конструкциях структурного программирования. Он предполагает пошаговую разработку алгоритма. Каждый шаг при этом включает разложение функции на подфункции. Так, на первом этапе приводят решение поставленной задачи, выделяя общие подзадачи. На следующем аналогично описывают решение подзадач, формулируя уже подзадачи следующего уровня. Таким образом, на каждом шаге происходит уточнение функций проектируемого ПО. Процесс продолжают, пока не доходят до подзадач, алгоритмы решения которых очевидны.

Декомпозируя программу методом пошаговой детализации, необходимо придерживаться основного правила структурной декомпозиции, следующего из принципа вертикального управления: в первую очередь детализировать управляющие процессы декомпозируемого компонента, оставляя уточнение операций с данными напоследок.

Кроме этого, целесообразно придерживаться следующих рекомендаций:

– не отделять операции инициализации и завершения от соответствующей обработки, так как модули инициализации и завершения имеют плохую связность (временную) и сильное сцепление (по управлению);

– не проектировать слишком специализированных или слишком универсальных модулей, так как проектирование излишне специальных модулей увеличивает их количество, а проектирование излишне универсальных – их сложность;

– избегать дублирования действий в различных модулях, так как при их изменении исправления придется вносить во все места, где они выполняются, – в этом случае целесообразно просто реализовать эти действия в отдельном модуле;

– группировать сообщения об ошибках в один модуль по типу библиотеки ресурсов, тогда будет легче согласовать формулировки, избежать дублирования сообщений, а также перевести сообщения на другой язык.

4.2. Объектно ориентированный подход при структурировании программного обеспечения

Определение логической структуры программного обеспечения (ПО) при объектно ориентированном подходе начинается с построения концептуальной модели предметной области.

Диаграммы классов – центральное звено объектно ориентированных методов разработки ПО, поэтому все существующие методы используют диаграммы классов в одной из известных нотаций. Однако в основном диаграммы классов в этих методах применяют на этапе проектирования для того, чтобы показать особенности построения конкретных классов.

В отличие от ранее существующих нотаций UML предлагает использовать три уровня диаграмм классов в зависимости от степени их детализации:

– концептуальный уровень, на котором диаграммы классов, называемые в этом случае контекстными, демонстрируют связи между основными понятиями предметной области;

– уровень спецификаций, на котором диаграммы классов отображают интерфейсы классов предметной области, т. е. связи объектов этих классов;

– уровень реализации, на котором диаграммы классов непосредственно показывают поля и методы конкретных классов.

Практически это три разных модели, связь между которыми неоднозначна. Так, если концептуальная модель определяет некоторое понятие предметной области как класс, то это не означает, что для реализации этого понятия будет использован отдельный класс. Однако во всех трех моделях нас интересуют типы объектов (классы) и их статические отношения, что позволяет использовать единую нотацию.

Каждая из перечисленных моделей используется на конкретном этапе разработки ПО:

– концептуальная – на этапе анализа;

– диаграммы классов уровня спецификации – на этапе проектирования;

– диаграммы классов уровня реализации – на этапе реализации.

Концептуальные модели в соответствии с определением оперируют: понятиями предметной области, атрибутами этих понятий и отношениями между ними. Понятию в предметной области разрабатываемого ПО могут соответствовать как материальные предметы, так и абстракции, которые применяют специалисты предметной области.

Основным понятиям в модели ставится в соответствие класс, который при этом традиционно понимают как совокупность общих признаков некоторой группы объектов предметной области. Согласно этому определению на диаграмме классов каждому классу соответствует группа объектов, общие признаки которых и фиксирует класс. Так, класс «Студент» объединя-

ет общие признаки группы людей, обучающихся в высших учебных заведениях. Экземпляр класса, или объект (например, Иванов И. И.), обязательно обладает всей совокупностью признаков своего класса и может иметь собственные признаки, не фиксированные в классе. Так, например, помимо того, что Иванов И. И. является студентом, он еще может быть спортсменом, музыкантом и т. д. Строго говоря, таким собственным признаком является и идентифицирующее студента имя.

На диаграммах класс изображается в виде прямоугольника, внутри которого отображается имя класса (рис. 8, *а*). При необходимости допускается указывать характеристики класса, например атрибуты, используя специальные секции условного обозначения (рис. 8, *б*).

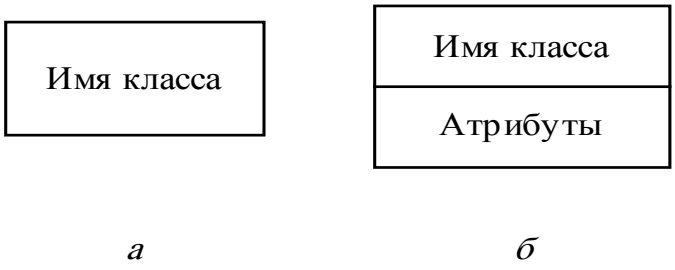

Рис. 8. Обозначение класса на концептуальной диаграмме классов:
а – без уточнения характеристик; *б* – с уточнением атрибутов

В качестве атрибутов представляют некоторые, существенные с точки зрения решаемой задачи, характеристики объектов, например, идентифицирующие значения (имя, номер). Для конкретного объекта атрибут всегда имеет конкретное значение. На диаграмме классов атрибуты обычно показывают в секции атрибутов.

Под отношением классов понимают статическую, т. е. не зависящую от времени, связь между классами. Различают два основных вида отношений: ассоциация и обобщение.

Отношение ассоциации означает наличие связи между экземплярами классов, или объектами: например, класс «студент» ассоциирован с классом «институт». Ассоциация может иметь имя: например, «Обучается». Рядом с именем ассоциации обычно ставят стрелку, указывающую направление чтения имени («Студент обучается в институте», а не наоборот).

Связь между экземплярами классов подразумевает некоторые роли, которые соответствующие объекты играют по отношению друг к другу. Роль связана с направлением ассоциации. Так, по отношению к студентам институт – организация, осуществляющая их обучение, т. е. роль института можно назвать «Место учебы». Студент для института – объект обучающей деятельности института, т. е. «Обучаемый». Если роль собственного имени не имеет, то можно считать, что ее имя совпадает с именем класса, по отношению к которому определяется эта роль. Для рассматриваемого примера это соответственно роли «Студент» и «Институт» (рис. 9, *а*), но роль можно указать и явно (рис. 9, *б*).

Роль также обладает характеристикой множественности, которая показывает, сколько объектов может участвовать в одной связи с каждой стороны (рис. 9, *в*). Допускается указывать множественность:

* – от 0 до бесконечности;

<целое>.. * – от заданного числа до бесконечности;

<целое> – точно определенное количество объектов;

<целое1>, <целое2> – несколько вариантов точного количества объектов;

<целое1>..<целое2> – диапазон объектов.

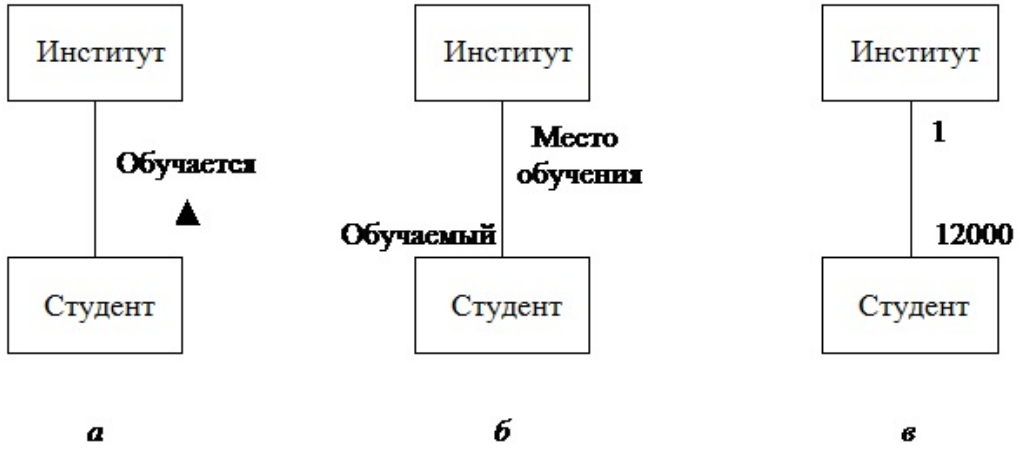

Рис. 9. Обозначение ассоциации: с указанием имени ассоциации (*а*); имен ролей (*б*); множественности (*в*)

С теоретической точки зрения атрибут тоже класс, экземпляры которого жестко ассоциированы с рассматриваемым классом. В качестве кон-

цептуальной модели для отображения соответствующих отношений могут использоваться как ассоциации, так и атрибуты. Например, отношение двух понятий «Студент» и «Имя» можно представить как в виде ассоциации соответствующих классов, так и в варианте, когда классу «Студент» ставится в соответствие атрибут «Имя».

Чтобы избежать излишних нагромождений рекомендуется следовать простому правилу: если некоторый объект X в реальном мире не является числом или текстом, то это скорее всего понятие. В противном случае – это атрибут.

Обобщением называют такое отношение между классами, при котором любой объект одного класса (подтипа) обязательно является также и объектом другого класса, называемого в данном контексте *супертипом*. На диаграмме классов обобщение обозначают линией с треугольной стрелкой на конце, подходящей к супертипу (рис. 10).

Рис. 10. Обозначение обобщения

На практике определение основных понятий предметной области, которые должны представляться на контекстной диаграмме в виде классов, является нетривиальной задачей. Обычно используют следующий способ:

1) формируют множество понятий-кандидатов из существительных, характеризующих предметную область в описании вариантов использования;

2) исключают понятия, не существенные для данного варианта использования.

Для определения множества понятий-кандидатов полезно использовать и перечень возможных категорий понятий-кандидатов.

Пример. Построить концептуальную объектно ориентированную модель для системы решения комбинаторно-оптимизационных задач.

Множество понятий-кандидатов для данной разработки включает: задание, тип задачи, список типов задач, способ задания данных, ввод данных, выбор данных из базы, алгоритм решения задачи, список конкретных алгоритмов решения задачи, полнота описания задания, результаты, данные, база данных.

Осуществим выделение основных понятий предметной области и свяжем их между собой.

Цель основного варианта использования системы – выполнение задания. Полное описание задания включает: тип задачи, данные и указание на алгоритм. С ним же будут связаны и полученные результаты. Данные могут сохраняться в базе и вводиться. Описание задания и все, что с ним связано, может сохраняться в базе.

Определим возможные обобщения:

1) способ задания данных: ввод данных, выбор данных из базы;

2) алгоритм: алгоритм решения задачи: конкретные алгоритмы решения задачи.

Переходим к построению концептуальной модели.

Основной класс-понятие, исходя из описания, – *Задание*. Связываем с ним классы-понятия *Данные, Алгоритм* и *Результаты*.

В разрабатываемой системе планируется реализовать алгоритмы решения задач трех типов: поиск цикла минимальной длины, проходящего через все вершины; поиск кратчайшего пути и поиск минимального покрывающего дерева. Следовательно, класс-понятие *Алгоритм* является супертипом для классов *Алгоритм поиска цикла минимальной длины, Алгоритм поиска кратчайшего пути* и *Алгоритм поиска минимального покрывающего дерева*, от которых, в свою очередь, будут наследоваться *Алгоритмы, реализующие конкретные методы*. *Алгоритм* также связан с *Данными* и *Результатами*.

Данные и *Задания* должны храниться в *Базе данных*, что показывают ассоциациями соответствующих классов. Способ задания данных для понимания основной концепции проектируемой системы пока не очень существен.

Вид задачи в нашем случае скорее атрибут класса *Задание*, чем самостоятельный класс, так как в реальном мире это имя, которое позволяет уточнить группу возможных алгоритмов решения и структуры исходных данных и получаемых результатов. Для алгоритма очень существенной характеристикой является его точность, соответственно добавим атрибут *Точность*. Другие атрибуты пока не проявились.

На рис. 11 показана полученная контекстная диаграмма классов программы решения комбинаторно-оптимизационных задач.

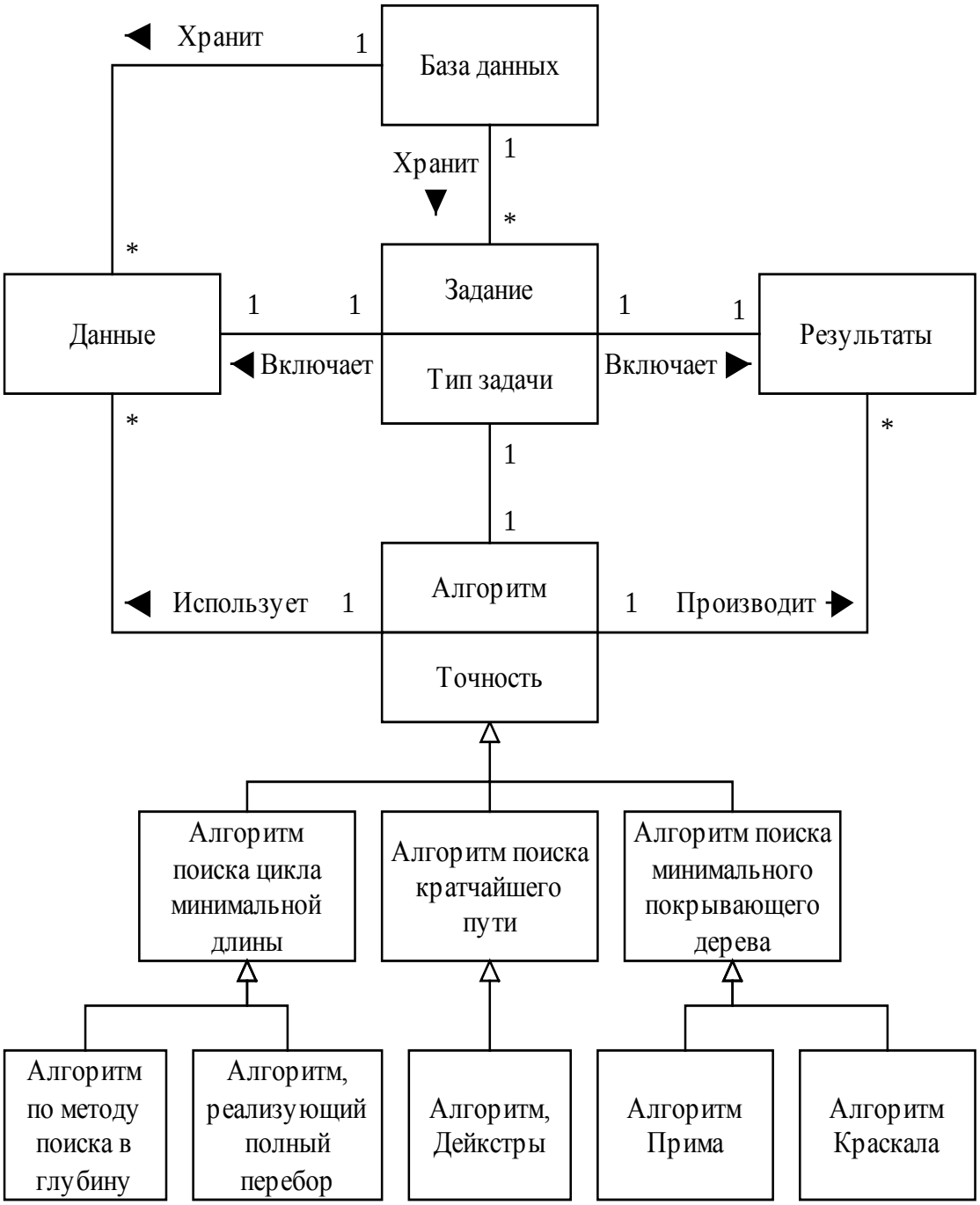

Рис. 11. Контекстная диаграмма классов

Концептуальная модель характеризует статические свойства разрабатываемого ПО. Для описания особенностей его поведения, т. е. возможных действий системы, целесообразно использовать диаграммы последовательностей системы, системные события, системные операции, диаграммы деятельностей, а при необходимости и диаграммы состояний объектов.

Множество всех системных операций определяют, идентифицируя системные события всех вариантов использования. Для наглядности системные операции изображают в виде операций абстрактного класса System (рис. 12).

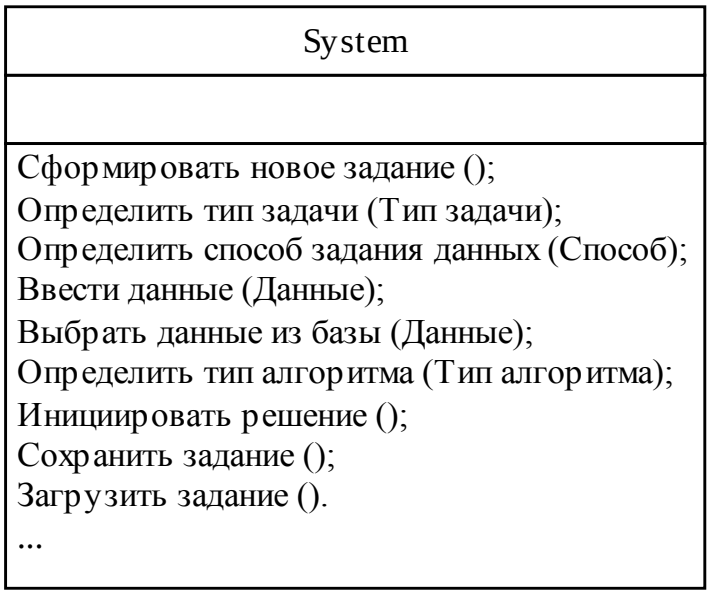

Рис. 12. Операции абстрактного класса

Каждую системную операцию необходимо описать. Обычно описание системной операции содержит:

– имя операции и ее параметры;

– описание обязанности;

– указание типа;

– названия вариантов использования;

– примечания для разработчиков алгоритмов и т. д.;

– описание обработки возможных исключений;

– описание вывода неинтерфейсных сообщений;

– предположение о состоянии системы до выполнения операции (предусловие);

– описание изменения состояния системы после выполнения операции (постусловие).

5. РАЗРАБОТКА АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

5.1. Описание алгоритмов

Алгоритм используется при написании любой программы и представляет собой пошаговую инструкцию выполнения вычислений или процесса. Очень часто решение той или иной задачи может быть реализовано с использованием различных алгоритмов. Например, простая задача нахождения по некоторому полю определенной записи в массиве записей может быть решена методом *последовательного поиска*, т. е. путем непрерывно следующего за другим считывания элементов массива, пока не будет найдена нужная запись или достигнут конец массива. Но можно применить и другие алгоритмы, например *бинарного поиска*. Если массив отсортирован по заданному полю, то поиск в этом случае осуществляется следующим образом: берется средний элемент массива, если это нужная запись, то поиск завершается, если нет, то определяют, где находится нужный элемент: в первой или второй половине – и повторяют операции, т. е. снова берут средний элемент в выбранной половине и т. д.

Для программы, использующей структурный подход к программированию, в данном разделе приводятся обобщенные алгоритмы, например, алгоритм основной программы, где описывается межмодульное взаимодействие подпрограмм, а также алгоритмы реализации отдельных процессов обработки информации или подпрограмм. Пример блок-схемы алгоритма приведен на рис. 13.

Для программы, при разработке которой использовалась объектно ориентированная технология, разрабатываемые алгоритмы должны соответствовать диаграммам классов и составу их операций. Для каждого класса желательно указать дополнительные поля и методы, обосновывая их назначение и функции. При необходимости здесь же можно привести алгоритмы некоторых методов.

Всего в пояснительной записке к РГР достаточно привести три-четыре наиболее интересных алгоритма, включая алгоритм основной программы при структурном подходе.

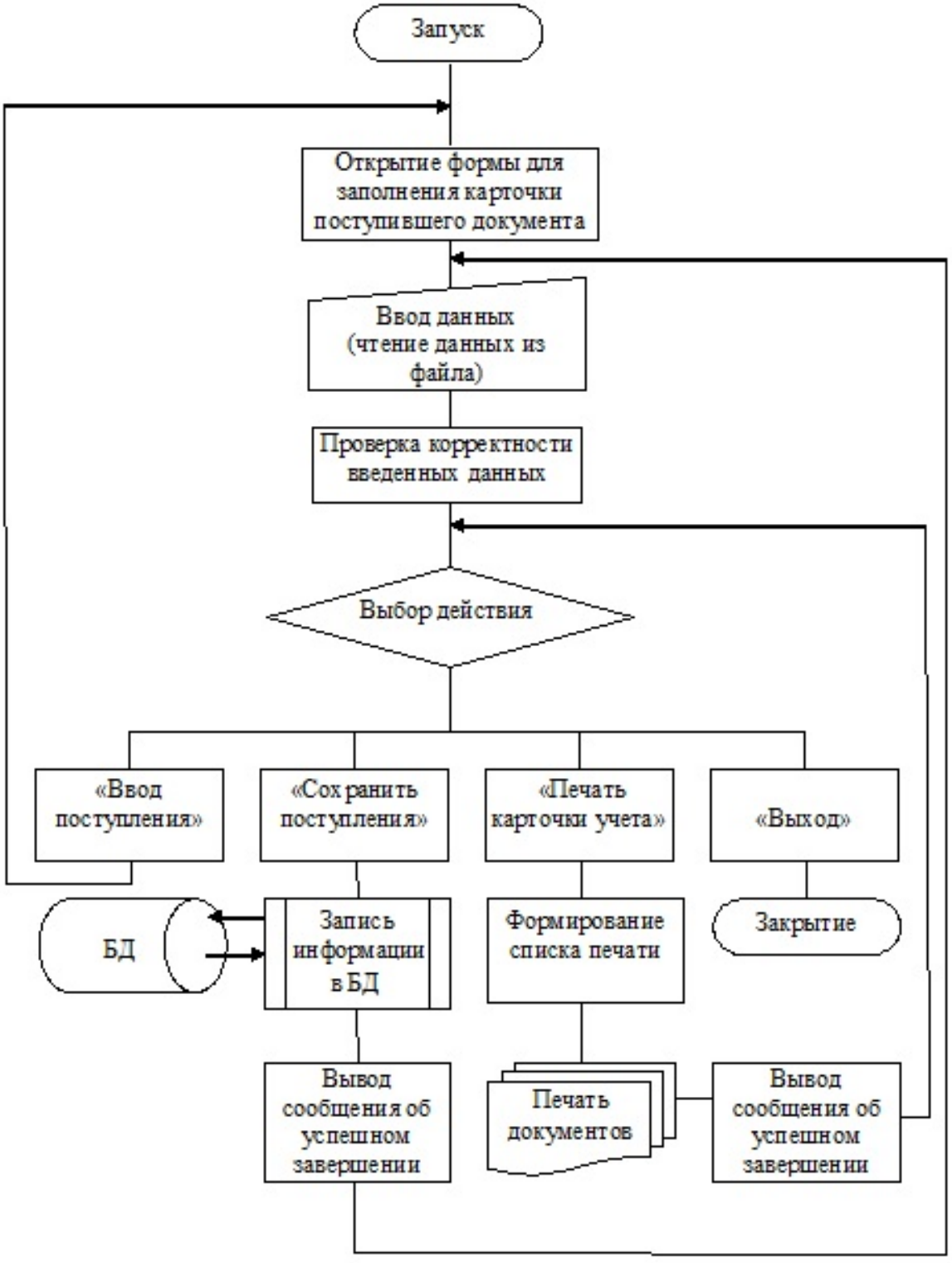

Рис.13. Алгоритм программного процесса

Описание каждого алгоритма должно включать:

- функциональное назначение алгоритма;
- входные и выходные данные (результаты выполнения);
- список формальных параметров и их назначение;

– пример вызова модуля или подпрограммы;
– используемые технические средства;
– ссылку на таблицу переменных алгоритма;
– ссылку на рисунок со схемой алгоритма;
– описание процесса обработки данных в соответствии со схемой.

При описании процесса обработки данных в соответствии со схемой алгоритма необходимо пояснить все циклы, каждую альтернативу ветвления, принятое решение по результатам анализа альтернатив и последующие действия.

Тексты описания алгоритмов должны быть структурными, предложения короткими. Описание алгоритма должно отражать суть процесса обработки. Кроме того, в этом же разделе желательно указать вариант разработки («восходящая» или «нисходящая») и обосновать свой выбор.

Основными критериями для выбора алгоритма являются его сложность, т. е. количество различных операций, выполняемых в процессе обработки, и эффективность, определяемая объемом памяти, необходимой для размещения данных, и его быстродействием. Быстродействие оценивается скоростью выполнения кода, а единственным методом оценки времени работы алгоритма является измерение времени его выполнения. Время выполнения кода в свою очередь отличается в зависимости от условий прохождения алгоритма и исходных данных.

Описания алгоритмов могут быть выполнены с использованием поведенческих типов диаграмм UML, таких как диаграммы деятельностей (*Activity Diagrams*), диаграммы последовательностей (*Sequence Diagrams*) и др.

5.2. Диаграммы деятельностей

Под *деятельностью* в данном случае понимают задачу (операцию), которую необходимо выполнить вручную или с помощью средств автоматизации. Каждому варианту использования соответствует своя последовательность задач. В теоретическом плане диаграммы деятельности – обобщенное представление алгоритма, реализующего анализируемый вариант использования.

На диаграмме деятельность обозначается прямоугольником с закругленными углами (рис. 14, *а*).

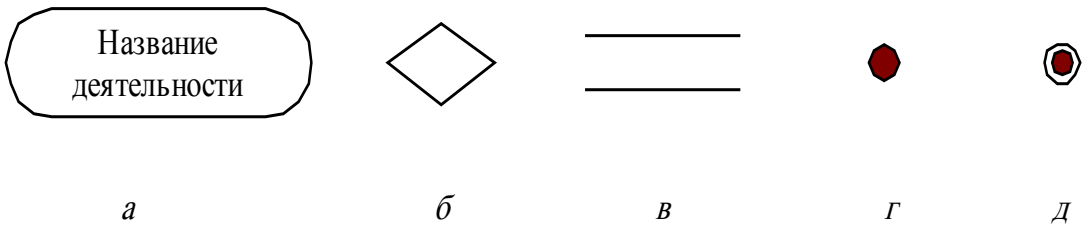

Рис. 14. Условные обозначения диаграммы деятельностей:
а – деятельность; *б* – выбор; *в* – линейки синхронизации; *г* – начало; *д* – конец

Диаграммы деятельности позволяют описывать альтернативные и параллельные процессы. Для обозначения альтернативных процессов используют ромб (рис. 14, *б*), условие указывают рядом, а альтернативы «да», «нет» – рядом с соответствующими выходами. С помощью этого же блока можно построить циклический процесс. Множественность активации деятельности обозначают символом «*», помещенным рядом со стрелкой активации деятельности, и при необходимости уточняют надписью вида «для каждой строки».

Для обозначения параллельных процессов используют линейки синхронизации (рис. 14, *в*), причем условие синхронизации можно уточнить, указав его на диаграмме. На рис. 15 показано, что «Деятельность 1» и «Деятельность 2» могут выполняться параллельно.

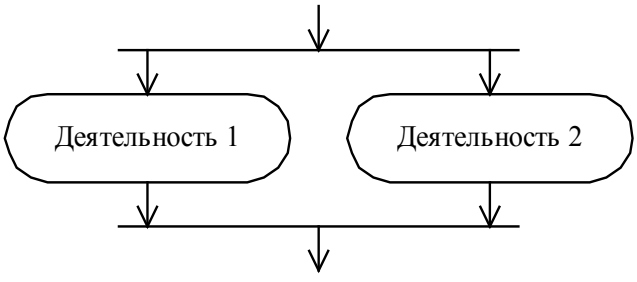

Рис. 15. Пример диаграммы деятельности с параллельными процессами

На этапе определения спецификаций имеет смысл уточнять только те варианты использования, краткого описания которых недостаточно для понимания сущности решаемых проблем.

Пример. Построить диаграмму деятельности, уточняющую вариант использования системы решения комбинаторно-оптимизационных задач.

Разбиваем процесс на операции, учитывая описание предметной области в виде контекстной диаграммы классов, и показываем их на диаграмме деятельности (рис. 16).

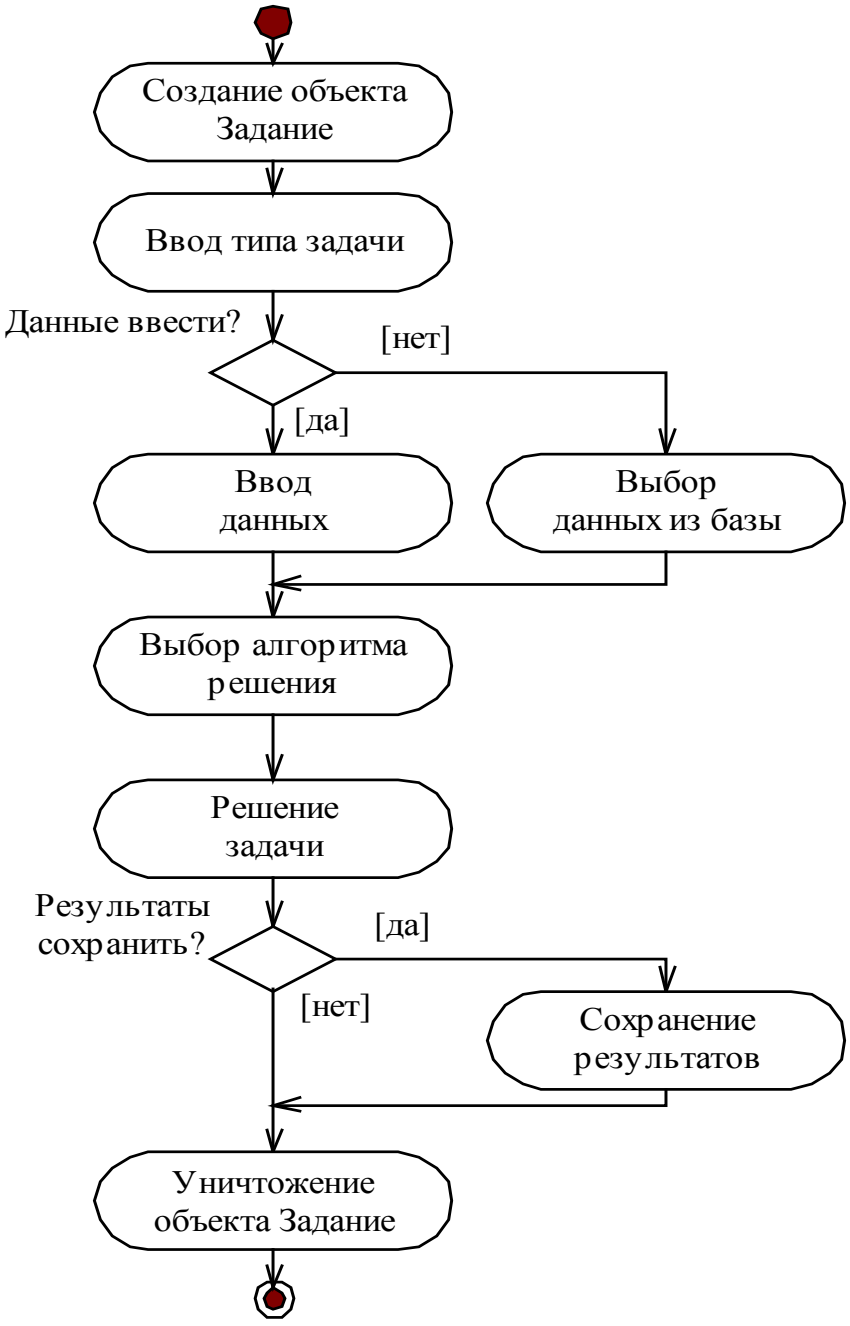

Рис. 16. Диаграмма деятельности, уточняющая вариант использования

Таким образом, диаграммы деятельностей можно использовать вместо описания вариантов использования или в дополнение к ним.

5.3. Диаграммы последовательностей

Диаграммы последовательностей этапа проектирования отображают взаимодействие объектов, упорядоченное по времени. В отличие от диаграмм последовательностей этапа анализа на ней показывают и внутренние объекты, а также последовательность сообщений, которыми обмениваются объекты в процессе реализации фрагмента варианта использования, обычно называемого *сценарием*.

Диаграммы последовательностей также позволяют изображать параллельные процессы. *Асинхронные* сообщения, которые не блокируют работу вызывающего объекта, показывают половинкой стрелки (рис. 17, *а*). Такие сообщения могут выполнять одну из трех функций:

– создавать новую ветвь процесса;
– создавать новый объект (рис. 17, *б*);
– устанавливать связь с уже выполняющейся ветвью процесса.

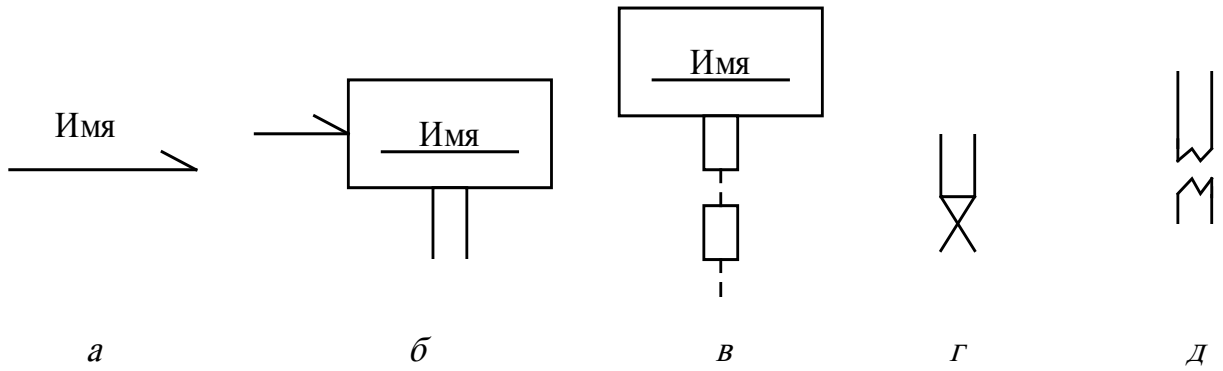

Рис. 17. Условные обозначения асинхронных передач управления:
а – асинхронное сообщение; *б* – создание объекта (не обязательно асинхронное);
в – активация объекта; *г* – уничтожение объекта; *д* – разрыв

На линии жизни для параллельных процессов дополнительно показывают *активации*, которые обозначаются прямоугольником, наложенным поверх линии жизни (рис. 17, *в*). Уничтожение объекта показывают большим знаком «X» (рис. 17, *г*). При необходимости линию жизни можно прервать, чтобы не уточнять обработку, не связанную с анализируемыми объектами (рис. 17, *д*).

Пример. Разработать диаграмму последовательностей для сценария Решение задачи.

Нормальный процесс предполагает, что при выдаче команды *Создать* создается объект *Решение*, управляющий данным сценарием. Следующее сообщение *Начать* активизирует этот объект. Объект *Решение* запрашивает у класса *Задание* тип объекта *Алгоритм*, создает объект требуемого класса и активизирует его, сохраняя способность получать и обрабатывать сообщения. Объект *Алгоритм* запрашивает у объекта класса *Задание* данные и начинает обработку, используя вспомогательные объекты. Нормально завершив обработку, объект класса *Алгоритм*, реализующий метод, передает объекту *Задание* результаты и возвращает объекту *Решение* признак нормального завершения. Объект *Решение* уничтожает объект класса *Алгоритм* и возвращает признак нормального завершения (рис. 18).

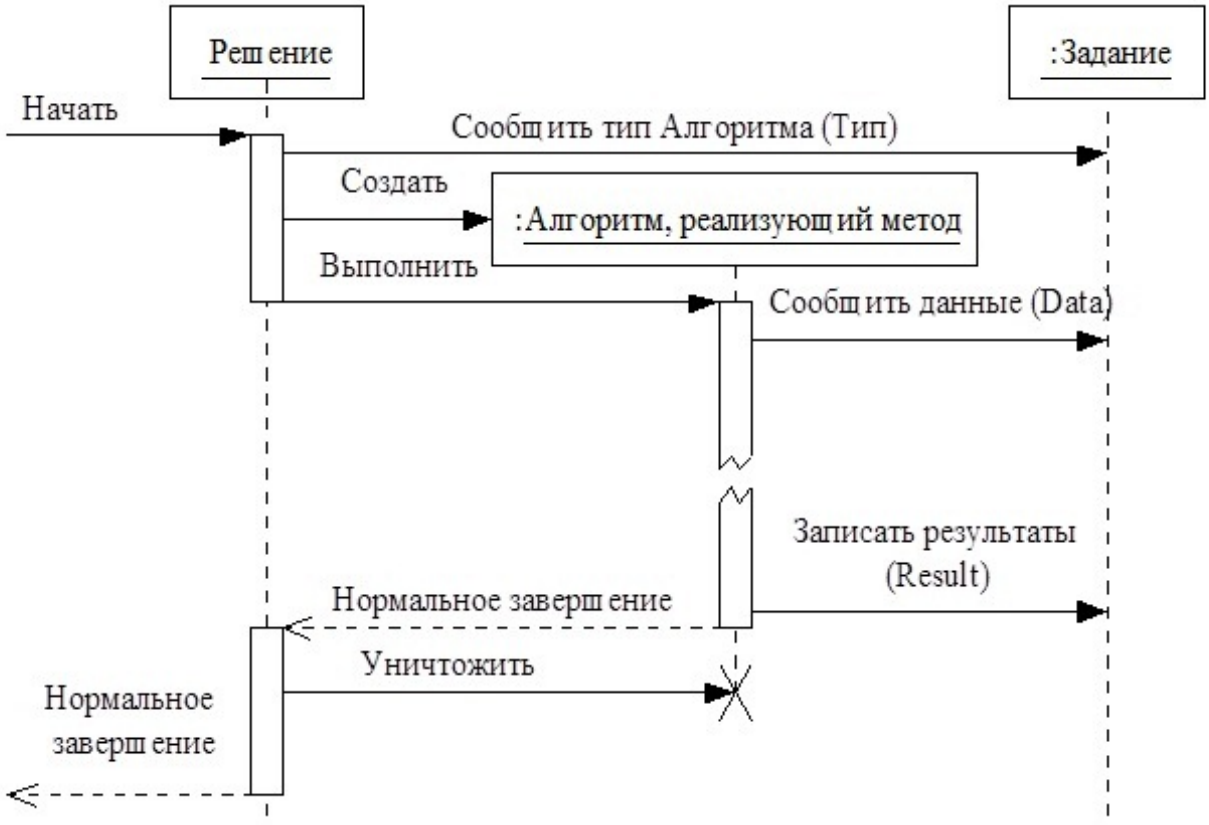

Рис. 18. Диаграмма последовательности действий процесса решения

В случае прерывания процесса объект *Решение* прерывает процесс решения, уничтожает объект *Алгоритм* и возвращает признак прерванного выполнения (рис. 19).

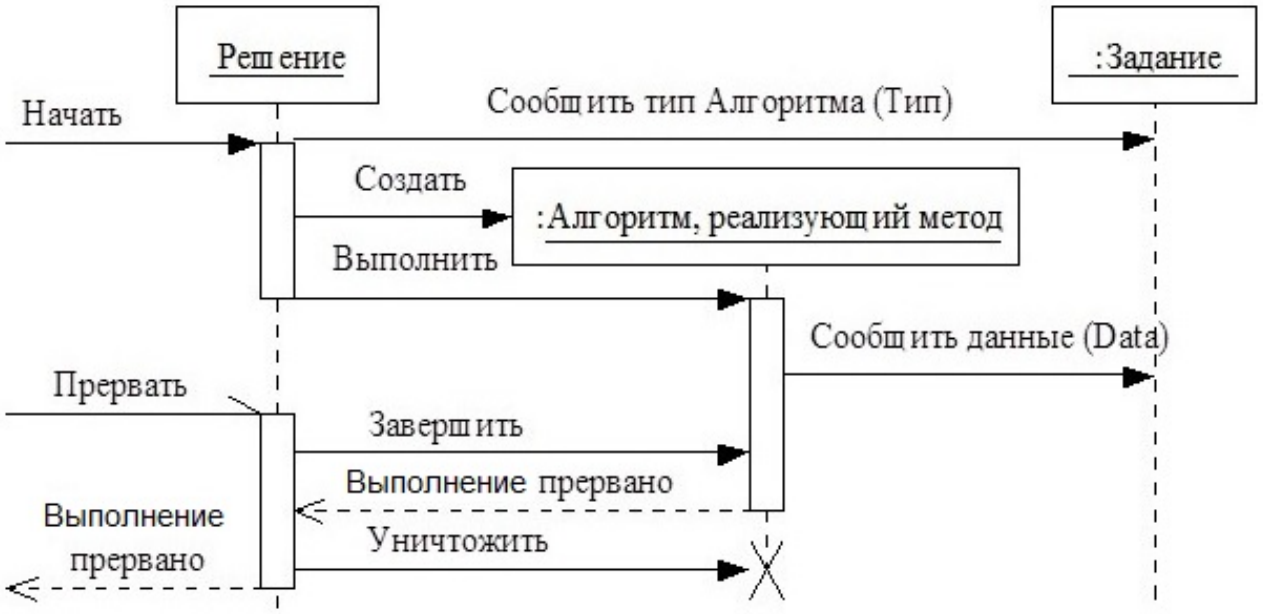

Рис.19. Диаграмма последовательности прерывания процесса пользователем

Анализ варианта использования данного сценария показывает, что необходимо рассмотреть такие варианты последовательности действий как нормальный процесс и прерывание процесса пользователем.

6. РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Интерфейсом пользователя – далее интерфейсом – будем называть совокупность способов и правил взаимодействия программы с пользователем.

В большинстве современных программных продуктов проектированию интерфейса уделяется огромное внимание. Даже такие программы, которые должны работать без вмешательства человека, например, управлять искусственным спутником Земли, снабжаются интерфейсом пользователя для обеспечения возможностей их установки, контроля и настройки. Исключением, пожалуй, являются встраиваемые микропрограммы, которые функционируют полностью автономно.

Можно без преувеличения сказать, что прекрасный во всех отношениях программный продукт, снабженный неудачным интерфейсом, не будет востребован на рынке ПО. Исключение составляют случаи естественного или искусственного монополизма, когда у пользователей нет выбора. Однако в настоящее время в любой сфере применения ПО монополизм любой программы ограничен весьма небольшим промежутком времени [6] .

Этот раздел содержит обзор различных способов и форм взаимодействия пользователя с системой и обоснование выбора определенной формы диалога (лежащего в основе любого взаимодействия) для общения с разрабатываемым программным продуктом. Проводится разработка структуры диалога и приводится граф диалога интерфейса, отражающий эту структуру. В случае табличной формы диалога дается описание всех оконных форм и меню (рис. 20 и 21).

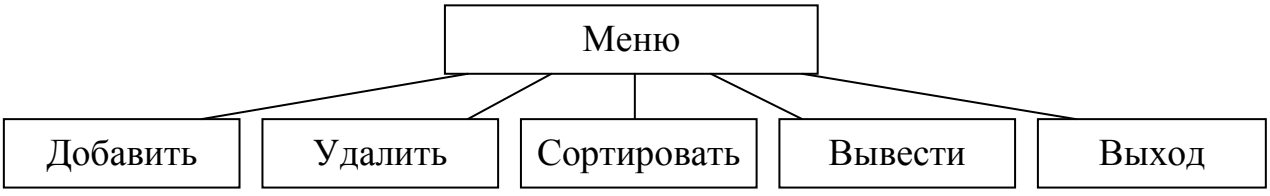

Рис. 20. Иерархии меню

В случае использования директивной или фразовой формы описываются основные команды.

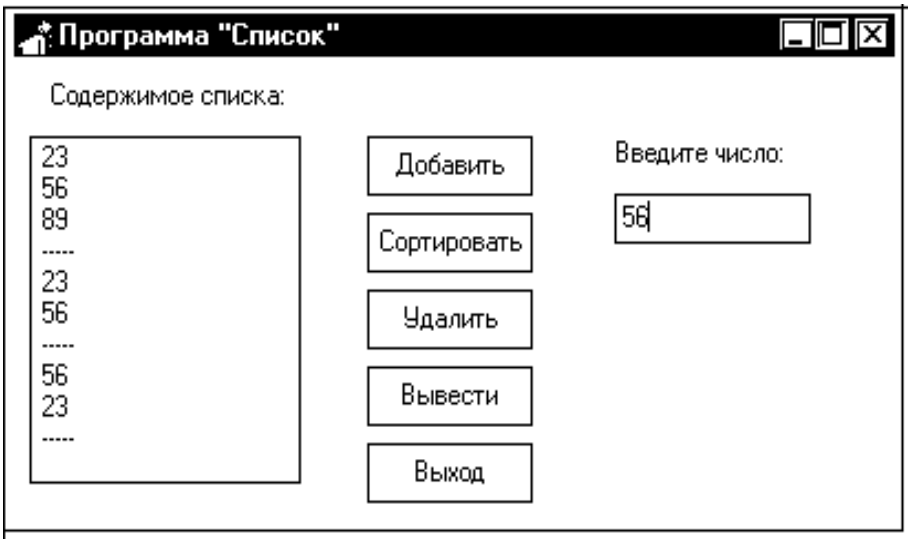

Рис. 21. Экранная форма

При использовании событийного программирования необходимо разработать и описать граф состояний интерфейса (рис. 22), на основе которого затем проектируются обработчики задействованных событий.

Рис. 22. Граф состояний интерфейса (при событийном программировании)

При проектировании интерфейса необходимо в первую очередь выбрать виды организации взаимодействия из перечисленных выше. При этом руководствуются следующими факторами. Во-первых, требование обеспечения развиваемости и модифицируемости программы при продолжающейся разработке предполагает отказ от жестко заданных примитивных схем диалога. Следует предусматривать возможности дальнейшего развития и усовершенствования продукта, что предполагает определенную гибкость в построении интерфейса. Во-вторых, финансовые и временные затраты на проектирование и разработку интерфейса должны быть обоснованы. Это означает, что в ряде случаев разрабатывать сложный и «продвинутый» интерфейс, затрачивая на это значительные ресурсы, нет реальной необходимости.

Для визуального интерфейса с использованием графических окон и элементов управления существует ряд общепринятых требований. Среди них можно выделить требования предсказуемости (интуитивной понятности), привлекательности, максимальной независимости от конкретных характеристик устройств ввода-вывода, возможности изменения настроек и целостности.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Басс, Л. Архитектура программного обеспечения на практике / Л. Басс, П. Клементе, Р. Кацман. – СПб. : Питер, 2006. – 384 с.

2. Белик, А. Г. Теория и технология программирования : учеб. пособие / А. Г. Белик, В. Н. Цыганенко. – Омск : Изд-во ОмГТУ, 2013. – 88 с.

3. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон. – М. : ДМК, 2006. – 496 с.

4. Гагарина, Л. Г. Технология разработки программного обеспечения / Л. Г. Гагарина, Е. В. Кокорев, Б. Д. Виснадул. – М. : Инфра-М, 2008. – 402 с.

5. Марка, Д. А. SADT – методология структурного анализа и проектирования / Д. А. Марка, К. М. Гоуэн. – М. : Метатехнология, 1993. – 231 с.

6. Тидвелл, Д. Разработка пользовательских интерфейсов / Д. Тидвелл. – СПб. : Питер, 2008. – 474 с.

7. Цыганенко, В. Н. CALS/CASE технологии проектирования информационных систем : учеб. пособие / В. Н. Цыганенко. – Омск : Изд-во ОмГТУ, 2007. – 88 с.

8. Цыганенко, В. Н. Технология программирования : метод. указания к курсовому проекту / В. Н. Цыганенко. – Омск : Изд-во ОмГТУ, 2005. – 44 с.

9. Цыганенко, В. Н. Технология программирования : учеб. пособие / В. Н. Цыганенко. – Омск : Изд-во ОмГТУ, 2005. – 32 с.