Министерство образования и науки РФ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ

Кафедра автоматизированных систем управления

С.Ю. Золотов

ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие

для студентов направления бакалавриата

09.03.03 «Прикладная информатика»

2016

Золотов С.Ю. Проектирование информационных систем: учебное пособие / С. Ю. Золотов. – Томск : 2016. – 117 с. [Электронный ресурс]

В учебном пособии раскрываются теоретические основы проектирования информационных систем. Особое внимание уделяется описанию структурного и объектно-ориентированного подходов к проектированию информационных систем.

Данное учебное пособие предназначено для студентов всех форм обучения направления бакалавриата 09.03.03 «Прикладная информатика».

Содержание

Введение

Без информационных систем уже немыслима жизнь современного человека. Информационные технологии прочно проникли во все сферы деятельности людей. Для разработки и поддержания в работоспособном состоянии информационных систем требуются знания об их проектировании.

Знания в области проектирования информационных систем включают в себя знания системного анализа прикладной области, формализации решения прикладных задач и процессов информационных систем (ИС), разработку требований к созданию и развитию ИС и ее компонентов, разработку проектов автоматизации и информатизации прикладных процессов и создание ИС в прикладных областях, а также управление проектами информатизации предприятий и организаций.

Данное учебное пособие предназначено для получения основных знаний по проектированию информационных систем. Для успешного усвоения дисциплины «Проектирование информационных систем» студент должен обладать знаниями системного анализа, общими сведениями об информационных технологиях, навыками программирования.

1. Общие сведения об информационных системах

1.1. Понятие информации

Прежде чем говорить об информации, необходимо дать определение энтропии [1].

Энтропия – это мера неопределенности какого-либо опыта, который может иметь разные исходы.

Пусть U – некоторый источник информации; N – количество событий, которые могут произойти; p_i – вероятность осуществления i-го события, причем $\sum_{i=1}^{N} p_i = 1$. В этом случае, величина энтропии рассчитывается по формуле:

$$H(U) = -\sum_{i=1}^{N} p_i \log_2 p_i .$$

Основные свойства энтропии:

1. Энтропия есть величина неотрицательна: $H \geq 0$.

2. Энтропия есть величина ограничена: $\lim_{p_i \to 0} p_i \log_2 p_i = 0$.

3. Энтропия равна нулю только тогда, когда одна из $p_i = 1$, а другие вероятности равны нулю.

4. Энтропия достигает своего максимального значения тогда, когда все события равновероятны.

5. Энтропия нескольких независимых источников равна сумме энтропии этих источников.

Принято говорить, что один объект отражает другой или содержит информацию о другом объекте, если состояние одного объекта находится в соответствии с состоянием другого объекта.

Информация есть свойство материи, состоящее в том, что в результате взаимодействия объектов между их состояниями устанавливается определенное соответствие. Чем сильнее выражено это соответствие, тем точнее состояние

одного объекта отражает состояние другого объекта, т.е. тем больше информации один объект содержит о другом.

В настоящее время информация рассматривается как фундаментальное свойство материи.

Сигнал – материальный носитель информации, т.е. средство переноса информации в пространстве и во времени. Один и тот же объект может выступать в роли разных сигналов. В качестве сигналов используются не сами объекты, а их состояния.

Сообщение – совокупность знаков или периодических сигналов, содержащих информацию. Одно и тоже сообщение ведет к разной интерпретации, т.е. к разной информации.

Количество информации I – это мера уменьшения энтропии объекта после совершения некоторого события.

Пусть H_0 – энтропия объекта до совершения события, H_1– энтропия объекта после совершения события, таким образом $H_0 > H_1 \Rightarrow I = H_0 - H_1$.

Единица информации – 1 бит (bit – BInery digiT).

Один бит – это количество информации, получаемое при осуществлении одного из двух равновероятных событий.

Основные производные единицы количества информации: 1 байт = 8 бит; 1Кбайт = 1024 байт; 1 Мбайт = 1024 Кбайт; 1 Гбайт = 1024 Мбайт; 1 Тбайт = 1024 Гбайт.

Вопросы для самоконтроля:

1. Что такое энтропия?
2. Что представляет понятие информации?
3. Чем отличается сигнал от сообщения?
4. Как рассчитать количество информации?
5. Чему равна минимальная единица количества информации?

1.2 Понятие информационных систем

Существует множество определений термина система [1, 2].

Первое определение системы:

Система есть средство достижения цели.

Другое определение:

Система – это совокупность взаимосвязанных элементов, обособленное от среды и взаимодействующая с ней как единое целое.

Можно дать и обобщенное определение:

Система – это конечное множество функциональных элементов и отношений между ними, выделяемое из среды в соответствии с определенной целью в рамках определенного временного интервала.

Элемент системы – некоторый объект, обладающий рядом важных свойств и реализующий в системе определенный закон функционирования, причем, внутренняя структура данного объекта не рассматривается.

Подсистема – это относительно независимая часть системы, которая обладает всеми свойствами системы и, в частности, имеет свою подцель, на достижение которой эта подсистема и ориентирована.

Если же части системы не обладают свойствами системы, а представляют собой просто совокупности однородных элементов, то такие части принято называть компонентами.

Под свойством понимают сторону объекта, обуславливающую его отличие от других объектов или сходство с ними и управляющуюся при взаимодействии с другими объектами.

Под управлением в самом общем виде будем понимать процесс формирования целенаправленного поведения системы посредством информационных воздействий, вырабатываемых человеком или устройством [2].

Система с управлением включает три подсистемы: управляющую систему (УС), объект управления (ОУ) и систему связи (СС). Управляющая система и система связи образуют систему управления (СУ). Основным элементом организационно-технической системы управления является лицо, принимающее решение (ЛПР).

Основными группами функций СУ являются [2]:

1. Функции принятия решений. Эти функции выражаются в создании новой информации в ходе анализа, планирования и оперативного управления. Это связано с преобразованием информации о состоянии ОУ и внешней среды в управляющую информацию при решении задач и выполнении аналитических расчетов, проводимых ЛПР при порождении и выборе альтернатив. Эта группа функций является главной, поскольку обеспечивает выработку информационных воздействий по удержанию в существующем положении или при переводе системы в новое состояние.

2. Функции обработки информации. Они охватывают учет, контроль, хранение, поиск, отображение, копирование информации. Эта группа функций по обработке информации не изменяет смысл самой информации (рутинные функции).

3. Функции обмена информацией. Эта группа функций связана с доведением выработанных воздействий ЛПР до ОУ.

Совокупность средств информационной техники и людей, объединенных для достижения определенных целей, в том числе и для управления, образуют информационную систему.

Информационная система (ИС) – это организационно-техническая система, использующая информационные технологии в целях обучения, информационно-аналитического обеспечения человеческой деятельности и процессов управления [2].

Вопросы для самоконтроля:

1. Что такое «система»?

2. Чем подсистема отличается от компоненты системы?
3. Что понимается под понятием «управление системой»?
4. Какие подсистемы входят в систему с управлением?
5. Какие бывают группы функций системы с управлением?
6. Что такое «информационная система»?

1.3 История развития информационных систем

Докомпьютерная эпоха (до 50-х гг. двадцатого века). До появления компьютеров ИС представляли собой различные архивы, картотеки и бухгалтерские книги. На карточке информация хранилась в упорядоченном виде, так же строго поддерживалась структурированная система закладок и каталогов для облегчения поиска информации. В больших информационных архивах поиск занимал много времени, а для поддержания строгой иерархической системы требовались значительные усилия [3].

Появление первых компьютеров (50-е и 60-е гг. двадцатого века). Появление первых компьютеров не оказало значительного влияния на мир ИС, так как первые компьютеры в основном занимались вычислениями и использовались преимущественно в военных целях. Тем не менее, у людей появилась возможность в эффективной автоматизированной обработки информации.

Первые коммерческие ИС (70-е гг. двадцатого века). По мере развития вычислительной техники в это время стали появляться первые коммерческие ИС. Они принадлежали крупным корпорациям и были довольно малочисленны. Это было связано с внушительной стоимостью как вычислительной техники, так и стоимостью создания самой ИС. Типичная ИС того времени работала на большой ЭВМ, к которой подсоединялись множество терминалов ввода данных. С терминалов посылали запросы на информацию и получали ответ на свои запросы. Большая ЭВМ обрабатывала запросы пакетами, и пользователям приходилось ожидать, пока не будет сформирован полный пакет для обработки.

Информация вводилась и выводилась в виде текстовой строки, что было неудобно для ее восприятия.

Информация централизованно хранилась в большой ЭВМ в виде базы данных. Каждая ИС требовала написания специализированного программного обеспечения и содержания штата высококвалифицированных специалистов для ее обслуживания. Первые ИС нашли свое применение в банковской сфере, на биржах и в крупных авиакомпаниях. Несмотря на огромную стоимость и неудобство использования, ИС доказали эффективность своего применения.

Появление персональных ЭВМ (с начала 80-х гг. двадцатого века)**.** Как было уже сказано, первые ИС обслуживали интересы крупных корпораций, причем общекорпоративные интересы или интересы руководства. Для конечного пользователя (особенно с увеличением количества подключенных терминалов) работа с ИС была не удобной и не помогала решать часть его задач, выбивавшихся из общекорпоративных шаблонов. Поэтому в начале 80-х годов появляются и быстро набирают популярность персональные ЭВМ. Они стоят гораздо дешевле большой ЭВМ и работают со стандартным программным обеспечением (ПО). В набор такого ПО входили текстовый редактор, электронные таблицы и персональная база данных, которую владелец мог настроить под свои нужды. Эти ЭВМ могли себе позволить уже значительно больший круг покупателей, что в дальнейшем привело к их стремительному развитию. Этот период можно назвать периодом популяризации автоматизированной обработки информации, и временем, когда компьютеры появились практически в каждом офисе.

Локальные сети (со второй половины 80-х гг. двадцатого века). Когда компьютеры появились на каждом рабочем месте, остро встала проблема обмена электронной информацией между сотрудниками одной организации. Решением этой проблемы стало объединение компьютеров одной организации в локальную вычислительную сеть. Сеть позволила сотрудникам обмениваться информацией не вставая с рабочего места и совместно использовать внешнее оборудование. Дальнейшее развитие локальных сетей стали локальные сети с вы-

деленным сервером – отдельным компьютером, предоставляющим какой-либо ресурс, в том числе и информацию, для всех участников сети. Такие компьютеры уже не использовались для обычных целей, они целиком специализировались на предоставлении ресурсов для совместного использования.

Распределенные ИС. Развитие локальных сетей с выделенным сервером привело к модели – один мощный сервер и несколько компьютеров-клиентов. Увеличение количества компьютеров-клиентов неизбежно вело к увеличению времени ожидания доступа к совместному ресурсу и конфликтов доступа между компьютерами-клиентами. Решением данной проблемы стало распределение ИС. Данное распределение означало разделение задачи серверного обслуживания на части между иерархической группой серверов, т.е. сервер нижнего уровня обслуживает несколько обычных компьютеров, сервера нижнего уровня обслуживаются сервером более высокого уровня и т.д. Таким образом, один сервер поддерживает небольшое количество клиентов, для обслуживания которых не требуется огромная мощность. При увеличении количества компьютеров-клиентов добавляется еще один сервер на соответствующий уровень, и нагрузка на каждый отдельный сервер этого уровня не увеличивается. Подобная организация ИС позволяет значительно удешевлять их стоимость и сравнительно просто наращивать мощность системы.

Глобализация ИС. К концу второго тысячелетия сформировалась тенденция глобализации информационных процессов. Этому способствовало широкое распространение глобальной сети Internet. Процессы глобализации обмена информацией увеличивают интенсивность, оперативность информационного обмена, обеспечивают мобильность сотрудников организации, позволяя работать в любое время суток с любого расстояния от офиса. Основными направлениями развития ИС стали разработка единого способа объединения информации разных форматов и совершенствование систем защиты информации от несанкционированного доступа.

Вопросы для самоконтроля:

1. Когда у людей появилась возможность в эффективной автоматизированной обработки информации?

2. В каких сферах бизнеса появились первые информационные системы?

3. Как проходила работа пользователей с информационными системами в 70-х годах 20 века?

4. Когда появились первые персональные компьютеры?

5. В чем была особенность программного обеспечения в момент появления персональных компьютеров?

6. В чем смысл появления локальных сетей?

7. Какой компьютер называется сервером?

8. Какую цель преследовала организация серверов в иерархические группы?

9. Что из себя представляет глобализация информационных процессов?

1.4 Характеристики современных информационных систем

Комплексный подход. Современные ИС характеризуются понятием комплексности. Это подразумевает целостный подход к автоматизации технологических процессов в организации. Если раньше в каждом отделе была своя, маленькая ИС, то сейчас вся организация работает в единой ИС. Такое построение ИС позволяет использовать информацию одного отдела в работе других подразделений, получать сводную информацию по всей организации и повышать скорость информационных потоков внутри организации [3].

Оперативность. В современных условиях очень важным параметром в работе организации становится скорость обработки и доступность информации. Поэтому современные ИС проектируются таким образом, чтобы пользователи могли получать максимум информации, доступной на текущий момент. Особое внимание уделяется оперативности информации, то есть процессам получения самой «свежей» информации, так как от этого во многом зависит эффектив-

ность принимаемых решений (например, в системах электронной торговли, в навигационных системах и т.д.).

Гибкость. Современные организации вынуждены работать в постоянно изменяющихся условиях, требующих изменений в структуре организации или методах ее работы. Поэтому важным параметром ИС является ее гибкость, т.е. способность быстро менять конфигурацию или функциональный набор. Наиболее распространенными способами реализации этого принципа являются модульность системы (при необходимости различные функциональные модули могут отключаться или подключаться к системе) и система настроек (т.е. заложена возможность коррекции основных параметров).

Распределенная ИС. Распределенная ИС подразумевает многоуровневую структуру и наличие иерархии серверов (см. раздел 1.3).

Взаимосвязь с другими ИС. Современная организация работает в условиях тесного взаимодействия и интенсивного информационного обмена с другими организациями, поэтому важное значение имеет способность «нашей» ИС взаимодействовать с ИС других организаций. В современных ИС должна быть предусмотрена хотя бы возможность импортировать и экспортировать массивы данных в общепринятых форматах обмена данными (текстовые файлы или электронные таблицы). Также необходима возможность взаимодействия системы с другими программными пакетами в рамках одной организации.

Доступность информации извне. В последнее время значительно увеличилась степень информационной открытости организаций для внешних потребителей (партнеров, клиентов). Современная ИС должна иметь механизмы публикации своих данных в Интернет для внешних пользователей (прайс-листы, перечень услуг, объявления). Естественно, не все данные организация делает общедоступными, поэтому большое внимание уделяется защите ИС от несанкционированного доступа и правильной организации уровней доступа к информации.

Вопросы для самоконтроля:

1. Что собой представляет комплексный подход в современных информационных системах?

2. Почему в современных информационных системах важное внимание уделяется оперативной информации?

3. Почему важным параметром информационной системы является ее гибкость?

4. Как современные информационные системы взаимодействуют между собой?

5. Зачем современные информационные системы имеют в своем составе сайты в глобальной сети?

1.5 Общая структура и состав информационной системы

Организационную структуру любого предприятия условно можно разделить на три части: руководство головного отделения, аналитический отдел и территориальные отделения. В головном отделении находится сервер базы данных (БД), куда стекается информация из всех территориальных отделений. Аналитический отдел работает со своим, отдельным сервером, на котором выполняются задачи этого отдела. Такая структура позволит разгрузить основной сервер от трудоемких задач аналитического отдела. Результаты работы аналитического отдела помещаются обратно на сервер данных, где их может увидеть руководство предприятия. Сервер базы данных ведет информационный обмен с серверами территориальных отделений. На рабочих местах специальное программное обеспечение формирует запросы к серверу приложений и представляет результаты, полученные от сервера БД.

ИС состоит из трех основных слоев: слой данных, слой приложений и слой интерфейса. Слой данных обеспечивает хранение данных и передачу данных по запросам слоя приложений. Слой приложений отвечает за обработку данных. Он взаимодействует со слоем данных и слоем интерфейса. Слой интерфейса

обеспечивает работу пользователей с графическими формами для формирования запросов и команд слою приложений и представления полученных результатов.

Каждый слой требует различный уровень характеристик компьютера. Для слоя данных необходим мощный компьютер с оптимизированной подсистемой работы с жестким диском (поскольку в основном его работа – запись и чтение данных) и скоростным сетевым адаптером для быстрой передачи данных по сети. Для ускорения операций ввода-вывода рекомендуется также повышенный объем оперативной памяти. Дополнительно требуется устройство резервного копирования для сохранения резервных копий данных.

Слой приложений требует компьютеров с мощным процессором и расширенной оперативной памятью, так как этого требует работа с приложениями. Рабочие места не требуют мощных компьютеров, для них достаточно конфигурации со стандартными характеристиками. Из дополнительного оборудования в этом слое чаще всего используется принтер.

1.6 Классификация информационных систем

В настоящий момент все существующие ИС можно разделить на четыре класса: корпоративные ИС, системы оперативного управления и учета, аналитические ИС, справочные правовые системы [3, 4].

Корпоративные ИС (КИС). КИС обеспечивают интегрированное решение задач управления предприятием как по вертикали (от первичной информации до поддержки принятия решений высшим руководством), так и по горизонтали (все направления деятельности и технологические операции). Весь спектр КИС можно разделить на три группы по степени интеграции: крупные, средние и малые. Они отличаются по набору функций, стоимости и сложности внедрения.

Крупные КИС чаще всего не являются готовыми ИС. Они представляют собой совокупность программных модулей и баз данных, а также технологии

их настройки и применения. Эти системы создаются как универсальные и многопрофильные, т.е. достаточно дорогие и сложные в установке и администрировании.

Оперативные ИС. Оперативная ИС предназначена для автоматизации оперативной деятельности организации, то есть для повышения эффективности ежедневной текущей деятельности сотрудников предприятия. В зависимости от рода текущей деятельности можно выделить четыре основных блока в оперативной ИС:

1) Блок документооборота. Учитываются входящие и исходящие документы как в бумажном виде, так и в электронном варианте. Блок включает фиксацию маршрута прохождения документа в организации, средства поиска документов и мониторинга их обработки, средства отчетности о документообороте в организации.

2) Блок учета ресурсов. В любой организации существует дефицит ресурсов – денежных, человеческих, транспортных средств, материалов и т.д. Оптимально и наиболее эффективно распределять эти ресурсы и призван данный блок. Он позволяет сотрудникам предприятия оперативно отслеживать состояние этих ресурсов и подавать заявки на их использование.

3) Блок бухгалтерского учета. Ведется полный бухгалтерский учет согласно существующим нормативным документам.

4) Блок кадрового учета. Ведет учет сотрудников организации, выполняет расчет зарплаты и премий, поддерживает штатное расписание и должностную иерархию предприятия. Данный блок позволяет вести мониторинг использования отпусков, больничных, начислять районные коэффициенты и надбавки за вредные условия работы.

Аналитические ИС (АИС). Основной целью АИС является накопление информации, необходимой для проведения полного и всестороннего анализа деятельности организации и среды ее функционирования. Для данного анализа не используется текущая информация, так как она часто меняется. Поэтому анализ

основывается на данных законченного временного периода: за прошлый месяц, прошлый квартал, прошлый год.

Структура АИС:

1) Блок накопления информации. Чаще всего информация поступает из оперативных ИС. Так как свойства оперативной и аналитической информации различны, то поступающую информацию необходимо преобразовать к виду, необходимому для АИС.

2) Блок поиска информации. За счет того, что данные в АИС агрегируются и упорядочиваются, система позволяет осуществлять более быстрый поиск в больших массивах данных при помощи множества индексов.

3) Блок анализа информации. Так как основной задачей АИС является собственно анализ информации, то система обладает достаточно развитыми средствами ранжирования информации, сравнительного анализа и проведения вычислений. Соответственно, поддерживается широкий набор встроенных статистических, математических и бизнес-функций.

4) Блок конфигурирования и настройки системы. С течением времени часто приходится менять структуру хранимых данных, поэтому АИС должны обладать гибкими средствами настройки. Такие средства включают в свой состав механизмы описания модели данных и дальнейшего использования этой модели при формировании запросов. Для оптимизации хранимых данных используются механизмы, позволяющие оценивать и анализировать качество построения данных с точки зрения скорости вычисления. При отсутствии оптимальности структура данных должна быть изменена.

Виды АИС:

1) Многомерные базы данных (МБД) (MDD – MultiDimensional Database). Информация в многомерной базе данных хранится не в виде индексированных записей в таблицах, а в форме многомерных упорядоченных массивов или кубов. Вид данных похож на данные в электронной таблице, только в таблице всего два измерения, а МБД-системе измерений может быть гораздо больше. Например, для анализа оказания услуг можно задать многомерный куб, где его

измерения будут делить информацию по дате оказания услуги, району отделения, по виду услуги, по исполняющей организации и т.д. Такой способ отображения информации получил название «звезда» (в центре данные, лучи – измерения). Этот способ хранения данных предпочтителен, когда необходима высокая скорость вычислений и выборок, требуются сложные вычисления, четко определена размерность задачи, невелико количество измерений, данные хорошо структурированы. Когда размерность задачи может или должна меняться, применение МБД нежелательно, так как изменение структуры данных связано с большой сложностью изменений. Также противопоказанием для применения МБД является множество невзаимосвязанных измерений.

2) Информационные хранилища (ИХ) (DW – Data Warehouse). В ИХ хранение информации похоже на хранение данных в МБД. Схема построения модели в ИХ получила название «снежинка». Здесь, в отличие от «звезды», каждый луч-измерение может иметь в свою очередь свои измерения. Такая схема позволяет более гибко формировать модель данных для сложных предметных областей. Плюсы ИХ заключается в более гибкой и рациональной модели данных, позволяющей описывать сложные предметные области, легкость модификации и дополнения модели данных.

3) Специализированные реляционные БД. Реляционная модель БД накладывает ограничения на скорость выборки и вычислений при сложной модели данных предметной области. Это ограничение послужило поводом для отказа от реляционной модели данных при хранении и обработке больших массивов информации. Позднее производители реляционных СУБД стали выпускать специально оптимизированные СУБД для аналитических данных, получивших название ROLAP-систем (Relation OnLine Analytical Processing). Такие продукты обладают специализированными средствами поиска и выборки данных, оптимизированных для аналитической обработки.

Справочно-правовые системы (СПС). В сфере юридической деятельности и правовой информатизации широко применяется термин «правовая информация». К правовой информации относятся правовые акты, материалы подготовки

законопроектов, комментарии и практика применения законов. С помощью СПС специалисту найти различные правовые акты и другие документы стало в сотни раз быстрее и эффективнее, чем при работе с бумажными носителями. У СПС есть две основных функции: 1) хранение и постоянное обновление базы правовой информации, отражающей текущее состояние действующего законодательства, 2) поиск документа в базе по неформально заданным критериям поиска.

Вопросы для самоконтроля:

1. Какие классы информационных систем существуют?

2. В чем особенность корпоративных информационных систем?

3. Почему оперативные информационные системы так называются?

4. Какие подсистемы есть в оперативной информационной системе?

5. В чем разница между аналитической информационной системой и оперативной?

6. Какие подсистемы есть в аналитической информационной системе?

7. В каком виде хранится информация в аналитической информационной системе?

2. Методологические основы проектирования информационных систем

2.1 Технология проектирования информационных систем

Современные информационные технологии предоставляют широкий выбор способов реализации ИС, выбор которых осуществляется на основе требований со стороны предполагаемых пользователей, которые, как правило, изменяются в процессе разработки. Для теории принятия решений процесс проектирования ИС – это процесс принятия проектно-конструкторских решений, направленных на получение проекта системы, удовлетворяющего требования заказчика [5].

Под проектом ИС будет понимать проектно-конструкторскую и технологическую документацию, в которой представлено описание проектных решений по созданию и эксплуатации ИС в конкретной программно-технической среде.

Под проектированием ИС понимается процесс преобразования входной информации об объекте проектирования, о методах проектирования и об опыте проектирования объектов аналогичного назначения в соответствии со стандартами в проект ИС.

С этой точки зрения проектирование ИС сводится к последовательной формализации проектных решений на различных стадиях жизненного цикла ИС.

Объектами проектирования ИС являются отдельные элементы или их компоненты функциональных и обеспечивающих частей. Так, функциональными элементами в соответствии с традиционной декомпозицией выступают задачи, комплексы задач и функции управления. В составе обеспечивающей части ИС объектами проектирования служат элементы и их компоненты информационного, программного и технического обеспечения системы.

В качестве субъекта проектирования ИС выступают коллективы специалистов, которые осуществляют проектную деятельность, как правило, в составе специализированной проектной организации, и организация-заказчик, для ко-

торой необходимо разработать ИС. При большом объеме и жестких сроках выполнения проектных работ в разработке системы может принимать участие несколько проектных коллективов. В этом случае выделяется головная организация, которая координирует деятельность всех организаций-соисполнителей.

Осуществление проектирования ИС предполагает использование проектировщиками определенной технологии проектирования, соответствующей масштабу и особенностям разрабатываемого проекта. Технология проектирования – это совокупность методологии и средств проектирования ИС, а также методов и средств организации проектирования.

В основе технологии проектирования лежит технологический процесс, который определяет действия, их последовательность, состав исполнителей, средства и ресурсы, требуемые для выполнения этих действий. Так, технологический процесс проектирования ИС в целом делится на совокупность последовательно-параллельных, связанных и соподчиненных цепочек действий. Действия, которые выполняются при проектировании ИС, могут быть определены как неделимые технологические операции или как подпроцессы технологических операций. Все действия могут быть собственно проектировочными, которые формируют или модифицируют результаты проектирования, и оценочными действиями, которые вырабатывают по установленным критериям оценки результатов проектирования.

Таким образом, технология проектирования задается регламентированной последовательностью технологических операций, выполняемых в процессе создания проекта на основе того или иного метода, в результате чего стало бы ясно, не только ЧТО должно быть сделано для создания проекта, но и КАК, КОМУ и в КАКОЙ ПОСЛЕДОВАТЕЛЬНОСТИ это должно быть сделано.

К основным требованиям, предъявляемым к выбираемой технологии проектирования, относятся следующие:

– созданный с помощью этой технологии проект должен отвечать требованиям заказчика;

– выбранная технология должна максимально отражать все этапы цикла жизни проекта;

– выбираемая технология должна обеспечивать минимальные трудовые и стоимостные затраты на проектирование и сопровождение проекта;

– технология должна быть основой связи между проектированием и сопровождением проекта;

– технология должна способствовать росту производительности труда проектировщика;

– технология должна обеспечивать надежность процесса проектирования и эксплуатации проекта;

– технология должна способствовать простому ведению проектной документации.

Основу технологии проектирования ИС составляет методология, которая определяет сущность, основные отличительные технологические особенности. Методология проектирования предполагает наличие некоторой концепции, принципов проектирования, реализуемых наборов методов проектирования, которые, в свою очередь, должны поддерживаться некоторыми средствами проектирования.

Организация проектирования предполагает определение методов взаимодействия проектировщиков между собой и с заказчиком в процессе создания проекта ИС, которые могут также поддерживаться набором специфических средств.

Для конкретных видов технологий проектирования свойственно применение определенных средств разработки ИС, которые поддерживают выполнение как отдельных проектных работ, этапов, так и их совокупностей. Поэтому перед разработчиками ИС, как правило, стоит задача выбора средств проектирования, которые по своим характеристикам в наибольшей степени соответствуют требованиям конкретного предприятия.

Средства проектирования должны быть:

– в своем классе инвариантными к объекту проектирования;

– охватывать в совокупности все этапы жизненного цикла ИС;

– технически, программно и информационно совместимыми;

– простыми в освоении и применении;

– экономически целесообразными.

Вопросы для самоконтроля:

1. Что такое проект информационной системы?

2. Что понимается под процессом проектирования информационной системы?

3. Что являются объектами проектирования информационной системы?

4. Кто выступает в качестве субъекта проектирования информационной системы?

5. Что такое технология проектирования?

6. Какие основные требования предъявляются к выбираемой технологии проектирования?

7. Что составляет основу технологии проектирования информационной системы?

2.2 Принципы проектирования сложных объектов

При проектировании сложных объектов используются следующие принципы [6]:

– декомпозиция и иерархичность построения описаний объектов проектирования;

– многоэтапность и итерационность процесса проектирования;

– типизация и унификация проектных решений.

Описания технических объектов должны быть по сложности согласованы: 1) с возможностями восприятия человеком; 2) с возможностями оперирования

описаниями в процессе их преобразования с помощью имеющихся средств проектирования.

Выполнить это требование в рамках единого описания удается лишь для простых изделий. Как правило, требуется структурирование описаний и соответствующее разбиение представлений об объекте на иерархические уровни и аспекты. Это позволяет распределить работы по проектированию сложных объектов между подразделениями проектировщиков, что способствует повышению эффективности и производительности труда.

Разделение описаний по степени детализации отображаемых свойств и характеристик объекта лежит в основе блочно-иерархического подхода к проектированию и приводит к появлению иерархических уровней (уровней абстрагирования) в представлениях об объекте.

На уровне 0 (верхнем уровне) сложный объект S рассматривается как система S из n взаимно связанных и взаимодействующих элементов S_i на уровне 1.

Каждый из элементов в описании уровня 1 представляет собой также довольно сложный объект, который, в свою очередь, рассматривается как описание системы S_i на уровне 2. Элементами системы S_i являются объекты $S_{i,j}$, $j = 1,2,\ldots,m_i$, где m_i – количество элементов в описании системы S_i. Как правило, выделение элементов $S_{i,j}$ происходит по функциональному признаку.

Подобное разбиение продолжается вплоть до получения на некотором уровне элементов, описания которых дальнейшему делению не подлежат, то есть до элементов, описание которых уже известно. Такие элементы по отношению к объекту S называются ***базовыми*** элементами.

Принцип иерархичности означает структурирование представлений об объекте проектирования по степени детальности описаний (уровни описаний – по вертикали)

Принцип декомпозиции (блочности) означает разбиение представлений каждого уровня на ряд составных частей (блоков) с возможностью раздельного

(поблочного) проектирования объектов S_i на уровне 1, объектов $S_{i,j}$ на уровне 2 и т.д. (каждый уровень разбивается на блоки.).

Кроме разбиения описаний по степени подробности отражения свойств объектов используют декомпозицию описаний по характеру отображаемых свойств объекта. Такая декомпозиция приводит к появлению ряда аспектов описаний. Наиболее крупные аспекты описаний объектов: функциональный; конструкторский; технологический. Решение задач, связанных с преобразованием или получением описаний, относящихся к этим аспектам, называют соответственно функциональным, конструкторским и технологическим проектированием.

Функциональный аспект связан с отображением основных принципов функционирования, характера физических и информационных процессов, протекающих в объекте. Функциональный аспект отображается в принципиальных, функциональных, структурных и других схемах и сопровождающих их документах.

Конструкторский аспект связан с реализацией результатов функционального проектирования, то есть с определением геометрических форм объектов и их взаиморасположением в пространстве.

Технологический аспект относится к реализации результатов функционального и конструкторского проектирования, т.е. связан с описанием методов и средств изготовления объектов.

Внутри каждого аспекта возможно свое специфическое выделение иерархических уровней.

Если решение задач высоких иерархических уровней предшествует решению задач более низких иерархических уровней, то проектирование называют *нисходящим*. Если раньше выполняются этапы, связанные с низшими иерархическими уровнями, то проектирование называют *восходящим*. У каждого из этих двух видов проектирования имеются преимущества и недостатки.

При нисходящем проектировании система разрабатывается в условиях, когда ее элементы еще не определены и, следовательно, сведения о их возможностях и свойствах носят предположительный характер.

При восходящем проектировании, наоборот, элементы проектируются раньше системы, и, следовательно, предположительный характер имеют требования к системе. В обоих случаях из-за отсутствия исчерпывающей исходной информации имеют место отклонения от возможных оптимальных технических результатов.

Поскольку принимаемые предположения могут не оправдаться, часто требуется повторное выполнение проектных процедур предыдущих этапов после выполнения проектных процедур последующих этапов. Такие повторения обеспечивают последовательное приближение к оптимальным результатам и обуславливают *итерационный* характер проектирования. Следовательно, итерационность нужно относить к важным принципам проектирования сложных объектов.

На практике обычно сочетают восходящее и нисходящее проектирование. Например, восходящее проектирование имеет место на всех тех иерархических уровнях, на которых используются *унифицированные* (стандартные) элементы. Очевидно, что унифицированные элементы, ориентированные на применение в ряде различных систем определенного класса, разрабатываются раньше, чем та или иная конкретная система из этого класса.

Обычно унификация объектов имеет целью улучшение технико-экономических показателей производства и эксплуатации изделий. Использование типовых и унифицированных проектных решений приводит так же к упрощению и ускорению проектирования.

Однако, унификация целесообразна только в таких классах объектов, в которых из сравнительно небольшого числа разновидностей элементов предстоит проектирование и изготовление большого числа систем. Именно эти разновидности элементов подлежат унификации.

Для сложных систем и для элементов, реализующих новые физические принципы или технологические возможности, в каждом конкретном случае приходится заново выполнять многоуровневое иерархическое проектирование.

В этих условиях целесообразно ставить вопрос не об унификации изделий, а об унификации средств их проектирования и изготовления, в частности об унификации проектных процедур в рамках систем автоматизированного проектирования (САПР).

Наличие средств автоматизированного выполнения типовых проектных процедур позволяет оперативно создавать проекты новых изделий, а в сочетании со средствами изготовления в условиях гибких автоматизированных производств осуществлять оперативное изготовление новых оригинальных изделий.

Окончательное описание проектируемого объекта представляет собой полный комплект схемной, конструкторской и технологической документации, оформленной в соответствии с требованиями ГОСТов: ЕСКД (единая система конструкторской документации), ЕСТД (единая система технологической документации), ЕСПД (единая система программной документации). Этот комплект документации предназначен для использования в процессе изготовления и эксплуатации объекта проектирования.

Важное значение в этих описаниях имеют математические модели объектов проектирования, так как выполнение проектных процедур при автоматизированном проектировании основано на оперировании математическими моделями.

Математическая модель (ММ) технического объекта – система математических объектов (чисел, переменных, матриц, множеств и т.п.) и отношений между ними, отражающих некоторые свойства технического объекта.

При проектировании используют математические модели, отражающие свойства объекта, существенные с позиции проектировщика.

Среди свойств объекта, отражаемых в описаниях на определенном иерархическом уровне, в том числе в ММ, различают свойства: систем; элементов систем и внешней среды, в которой должен функционировать объект. Количе-

ственное выражение этих свойств осуществляется с помощью величин, называемых параметрами. Величины, характеризующие свойства системы, элементов системы и внешней среды, называют соответственно входными, внутренними и внешними параметрами.

Однако существование ММ не означает, что она известна разработчику и может быть представлена в явном функциональном виде. Типичной является ситуация, когда математическое описание процессов в проектируемом объекте задается моделью в форме системы уравнений, в которой фигурирует вектор фазовых переменных. Фазовые переменные характеризуют физическое или информационное состояние объекта, а их изменения во времени выражают переходные процессы в объекте. Например, состояние некоторой фирмы можно определить такими фазовыми переменными: сырье, материалы, финансовые и трудовые ресурсы.

Выделим следующие особенности параметров в моделях проектируемых объектов:

1. Внутренние параметры в моделях k-го иерархического уровня становятся выходными параметрами в моделях более низкого $(k+1)$-го иерархического уровня. Так, например, трудовые ресурсы являются внутренними при проектировании производственной фирмы и в то же время выходными при проектировании отдела кадров этой фирмы.

2. Выходные параметры или фазовые переменные, фигурирующие в модели одной из подсистем (в одном из аспектов описаний), часто оказываются внешними параметрами в описании других подсистем (других аспектов). Так, например, выходные параметры подсистемы планирования выпуска продукции некоторой компании являются внешними параметрами подсистемы материально-технического снабжения этой компании.

3. Большинство выходных параметров объекта являются функционалами.

4. В техническом задании на проектирование должны фигурировать величины, называемые техническими требованиями к выходным параметрам (нор-

мами выходных параметров). Данные нормы представляют собой границы допустимых диапазонов изменения выходных параметров.

Вопросы для самоконтроля:

1. Какие принципы используются при проектировании сложных объектов?
2. Какие элементы называются базовыми?
3. Какое проектирование называют нисходящим, а какое – восходящим?
4. Что представляет собой итерационный характер проектирования?
5. Какие элементы называются унифицированными?
6. Что такое математическая модель технического объекта?
7. Какие величины называются параметрами модели?
8. Какие переменные называются фазовыми в моделях объекта?
9. В чем заключаются особенности параметров в моделях проектируемых объектов?

2.3 Классификация типовых проектных процедур

Проектная процедура называется *типовой*, если она предназначена для многократного применения при проектировании многих типов объектов.

Различают проектные процедуры анализа и синтеза. Синтез заключается в создании описания объекта, а анализ – в определении свойств и исследовании работоспособности объекта по его описанию, т.е. при синтезе создаются, а при анализе оцениваются проекты объектов [6].

Процедуры анализа делятся на процедуры одно- и многовариантного анализа. При одновариантном анализе заданы значения внутренних и внешних параметров, требуется определить значения выходных параметров объекта. Подобная задача обычно сводится к однократному решению уравнений, составляющих математическую модель, что и обуславливает название этого вида анализа. Многовариантный анализ заключается в исследовании свойств объекта в некоторой области пространства внутренних переменных. Такой анализ требу-

ет многократного решения систем уравнений (многократного выполнения одновариантного анализа).

Процедуры синтеза делятся на процедуры структурного и параметрического синтеза. Целью структурного синтеза является определение структуры объекта – перечня типов элементов, составляющих объект, и способа связи элементов между собой в составе объекта. Параметрический синтез заключается в определении числовых значений параметров элементов при заданных структуре и условиях работоспособности на выходные параметры объекта, т.е. при параметрическом синтезе нужно найти точку или область в пространстве внутренних параметров, в которых выполняются те или иные условия (обычно условия работоспособности).

На рис. 2.1 представлена типичная последовательность проектных процедур на одном из этапов нисходящего проектирования. На предыдущем этапе решались задачи $(k-1)$-го иерархического уровня.

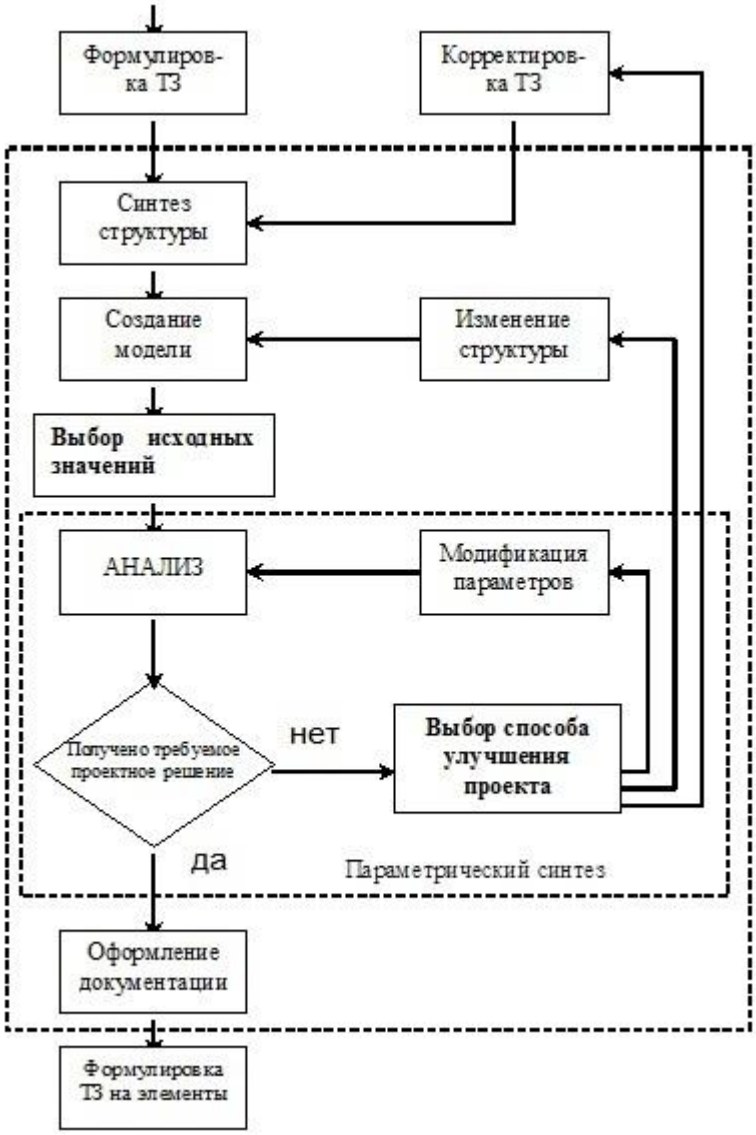

Рис. 2.1 – Схема процесса проектирования

Проектирование системы начинается с синтеза исходного варианта ее структуры. Для оценки этого варианта создается: математическая модель (при автоматизированном проектировании); экспериментальная модель или стенд (при неавтоматизированном проектировании).

После выбора исходных значений параметров элементов выполняется анализ варианта, по результатам которого становится возможной его оценка. Обычно оценка заключается в проверке выполнения условий работоспособности, сформулированных в техническом задании (ТЗ).

Если условия работоспособности выполняются в должной мере, то полученное проектное решение принимается, затем описывается система $(k+1)$-го

уровня в принятой форме и формулируется ТЗ на проектирование элементов данного уровня. Если же полученное проектное решение неудовлетворительно, то выбирается один из возможных путей улучшения проекта.

Обычно проще всего изменить числовые значения параметров элементов. Совокупность процедур модификации параметров элементов, анализа и оценки результатов анализа представляет собой процедуру *параметрического синтеза*. Если модификации параметров целенаправленны и подчинены стратегии поиска наилучшего значения некоторого показателя качества, то процедура параметрического синтеза является *процедурой оптимизации*.

Возможно, что путем параметрического синтеза не удастся добиться приемлемой степени выполнения условий работоспособности. Тогда используют другой путь, связанный с модификациями структуры. Новый вариант структуры синтезируется, для него повторяются процедуры формирования модели и параметрического синтеза.

Если не удается получить приемлемое решение и на этом пути, то ставится вопрос о корректировке ТЗ, сформулированного на предыдущем этапе проектирования. Такая корректировка может потребовать повторного выполнения ряда процедур k-го иерархического уровня, что и обуславливает итерационный характер проектирования.

Существует характерная особенность взаимосвязи проектных процедур анализа и синтеза. Эта взаимосвязь имеет характер вложенности процедуры анализа в процедуру оптимизации (параметрического синтеза) и процедуры оптимизации в процедуру синтеза (структурного и параметрического).

Вложенность означает, что:

– анализ входит как составная часть в оптимизацию, а оптимизация – в синтез;

– однократное выполнение процедуры оптимизации требует многократного выполнения процедуры анализа;

– однократное решение задачи синтеза требует многократного решения задачи оптимизации.

Очевидно, такой же характер взаимодействия имеют процедуры анализа: однократный многовариантный анализ основан на многократном одновариантном анализе.

Нетрудно подсчитать, что синтез проектного решения может потребовать чрезмерно большого количества вариантов анализа. Один из путей решения этой проблемы – применение достаточно точных и сложных математических моделей и алгоритмов анализа только на завершающих итерациях синтеза. Для большинства просматриваемых вариантов структуры при этом выполняется лишь ориентировочная оценка на основе косвенных критериев, упрощенных моделей и алгоритмов. Такая оценка позволит без существенных затрат вычислительных ресурсов отсеять большинство неперспективных вариантов и оставить для тщательного анализа малое число вариантов.

Вопросы для самоконтроля:

1. Какая проектная процедура называется типовой?
2. Какие бывают проектные процедуры?
3. На какие процедуры делятся процедуры анализа?
4. В чем смысл процедуры одновариантного анализа?
5. В чем смысл процедуры многовариантного анализа?
6. На какие процедуры делятся процедуры синтеза?
7. В чем смысл процедуры структурного синтеза?
7. В чем смысл процедуры параметрического синтеза?
8. Какая процедура называется процедурой оптимизации?

2.4 Жизненный цикл информационной системы

Проектирование ИС – трудоемкий, длительный и динамический процесс. Технологии проектирования, применяемые в настоящее время, предполагают поэтапную разработку системы. Этапы по общности целей могут объединяться в стадии.

Совокупность стадий и этапов, которые проходит ИС в своем развитии от момента принятия решения о создании системы до момента прекращения функционирования системы, называется *жизненным циклом ИС* [5].

Суть содержания жизненного цикла (ЖЦ) разработки ИС в различных подходах одинакова и сводится к выполнению следующих стадий:

1. Стадия планирования и анализа требований (предпроектная стадия системного анализа): исследование и анализ существующей ИС, определение требований к создаваемой ИС, оформление технико-экономического обоснования и технического задания на разработку ИС.

2. Стадия проектирования: разработка в соответствии со сформулированными требованиями состава автоматизируемых функций и состава обеспечивающих подсистем, оформление технического проекта ИС.

3. Стадия реализации: разработка и настройка программ, наполнение базы данных, создание рабочих инструкций для персонала.

4. Стадия внедрения: комплексная отладка подсистем ИС, обучение персонала, поэтапное внедрение ИС в эксплуатацию по подразделениям предприятия, оформление акта о приемо-сдаточных испытаниях ИС.

5. Стадия эксплуатации ИС: сбор рекламаций и статистики о функционировании ИС, исправление ошибок и недоработок, оформление требований к модернизации ИС и их выполнение (повторение стадий 2–5).

Важной чертой жизненного цикла ИС является его повторяемость «Планирование и анализ требований – разработка – сопровождение – планирование и анализ требований». Это соответствует представлению об ИС как о развивающейся, динамической системе. При первом выполнении стадии разработки создается проект ИС, а при повторном выполнении осуществляется модификация проекта для поддержания его в актуальном состоянии.

Другой характерной чертой жизненного цикла является наличие нескольких серий его прохождения:

1. Первая серия – это серия первичного проектирования ИС.

2. Вторая серия – это серия, которая возникает после опытного внедрения, в результате которого выясняются частные ошибки в элементах проекта.

3. Третья серия возникает после сдачи в промышленную эксплуатацию, когда выявляют ошибки в функциональной архитектуре системы, связанные с несоответствием проекта требованиям заказчика по составу функциональных подсистем, составу задач и связям между ними.

4. Четвертая серия возникает в том случае, когда требуется модификация системной архитектуры в связи с необходимостью адаптации проекта к новым условиям функционирования системы.

5. Пятая серия возникает, если проект системы совершенно не соответствует требованиям, предъявляемые к организационно-экономической системе ввиду того, что осуществляется моральное его старение.

Чтобы исключить пятую серию и максимально уменьшить необходимость выполнение третьей и четвертой серий, необходимо выполнять проектирование ИС на всех этапах первой, основной серии разработки ИС, в соответствии со следующими требованиями:

– разработка ИС должна быть выполнена в строгом соответствии со сформулированными требованиями к создаваемой системе;

– требования к ИС должны адекватно соответствовать целям и задачам эффективного функционирования самого объекта;

– созданная ИС должна соответствовать сформулированным требованиям на момент окончания внедрения, а не на момент начала разработки;

– внедренная ИС должна развиваться и адаптироваться в соответствии с постоянно изменяющимися требованиями к ИС.

С точки зрения реализации перечисленных аспектов в технологиях проектирования ИС модели жизненного цикла претерпевали существенные изменения. Среди известных моделей жизненного цикла можно выделить следующие модели:

– каскадная модель (до 70-х годов) – последовательный переход на следующий этап после завершения предыдущего;

– итерационная модель (70 – 80-е годы) – с итерационными возвратами на предыдущие этапы после выполнения очередного этапа;

– спиральная модель (современное время) – модель, предполагающая постепенное расширение прототипа ИС.

Каскадная модель. Для этой модели жизненного цикла характерна автоматизация отдельных несвязанных задач, не требующая выполнения информационной интеграции и совместимости, программного, технического и организационного сопряжения. В рамках решения отдельных задач каскадная модель жизненного цикла по срокам разработки и надежности оправдала себя. Применение каскадной модели жизненного цикла к большим и сложным проектам вследствие большой длительности процесса проектирования и изменчивости требований за это время приводит к их практической нереализуемости.

Итерационная модель. Создание комплексных ИС предполагает проведение увязки проектных решений, получаемых при реализации отдельных задач. Подход к проектированию «снизу-вверх» обуславливает необходимость таких итерационных возвратов, когда проектные решения по отдельным задачам комплектуются в общие системные решения и при этом возникает потребность в пересмотре ранее сформулированных требований. Как правило, вследствие большого числа итераций возникают рассогласования в выполненных проектных решениях и документации. Запутанность функциональной и системной архитектуры созданной ИС, трудность в использовании проектной документации вызывают на стадиях внедрения и эксплуатации сразу необходимость перепроектирования всей системы. Длительный жизненный цикл разработки ИС заканчивается этапом внедрения, за которым начинается жизненный цикл создания новой ИС.

Спиральная модель. Используется подход к организации проектирования ИС «сверху-вниз», когда сначала определяется состав функциональных подсистем, а затем постановка отдельных задач. Соответственно, сначала разрабатываются такие общесистемные вопросы, как организация интегрированной базы данных, технология сбора, передачи и накопления информации, а затем техно-

логия решения конкретных задач. В рамках комплексов задач программирование осуществляется по направлению от головных программных модулей к исполняющим отдельные функции модулям. При этом на первый план выходят вопросы взаимодействия интерфейсов программных модулей между собой и с базой данных, а на второй план – реализация алгоритмов.

Вопросы для самоконтроля:

1. Что называется жизненным циклом информационной системы?

2. Из каких стадий состоит жизненный цикл информационной системы?

3. Какие требования нужно учесть при разработке информационной системы?

4. Какие существуют модели жизненного цикла?

3 Структурный подход к проектированию информационных систем

3.1 Сущность структурного подхода

Сущность структурного подхода к разработке ИС заключается в ее декомпозиции на автоматизируемые функции: т.е. система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, эти подфункции подразделяются на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны [6].

Все наиболее распространенные методологии структурного подхода базируются на ряде общих принципов:

– принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;

– принцип иерархического упорядочивания, т.е. принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

– принцип абстрагирования: заключается в выделении существенных аспектов системы и отвлечения от несущественных;

– принцип формализации: заключается в необходимости строгого методического подхода к решению проблемы;

– принцип непротиворечивости: заключается в обоснованности и согласованности элементов;

– принцип структурирования данных: заключается в том, что данные должны быть структурированы и иерархически организованы.

На стадии проектирования ИС модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру ПО, структурные схемы программ и диаграммы экранных форм.

Перечисленные модели в совокупности дают полное описание ИС независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

3.2 Методология функционального моделирования SADT

Методология SADT (Structured Analysis and Design Technique) представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями [6].

Дальнейшим развитием данной методологии стало появление стандарта IDEF0. IDEF (Integration DEFinition) – семейство совместно используемых методов для процесса моделирования [8].

Результатом применения методологии SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы - главные компоненты модели, все функции ИС и интерфейсы на них представлены как блоки и дуги. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, показывается с левой стороны блока, а результаты выхода – с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу (рис. 3.1).

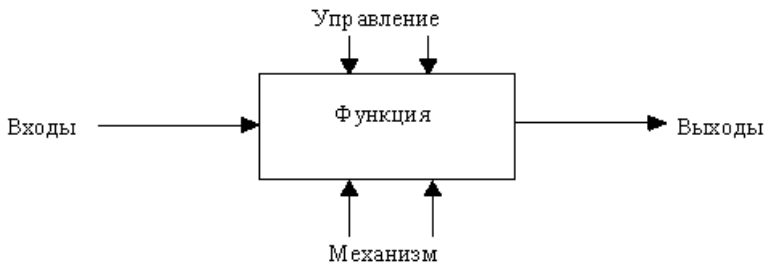

Рис. 3.1 – Функциональный блок и интерфейсные дуги

SADT-модель дает полное, точное и адекватное описание системы, имеющее конкретное назначение. Целью модели является получение ответов на некоторую совокупность вопросов. Эти вопросы неявно присутствуют (подразумеваются) в процессе анализа и, следовательно, они руководят созданием модели и направляют его. Это означает, что сама модель должна будет дать ответы на эти вопросы с заданной степенью точности. Если модель отвечает не на все вопросы или ее ответы недостаточно точны, то мы говорим, что модель не достигла своей цели. Определяя модель таким образом, SADT закладывает основы практического моделирования. Только поняв, насколько хорошо нужно ответить на поставленные вопросы, можно определить, когда процесс моделирования можно считать завершенным (т.е. когда модель будет соответствовать поставленной цели).

Модель является некоторым толкованием системы. Поэтому субъектом моделирования служит сама система. Однако моделируемая система никогда не существует изолированно: она всегда связана с окружающей средой. Причем зачастую трудно сказать, где кончается система и начинается среда. По этой причине в методологии SADT подчеркивается необходимость точного определения границ системы. SADT-модель всегда ограничивает свой субъект, т.е. модель устанавливает точно, что является и что не является субъектом моделирования, описывая то, что входит в систему, и подразумевая то, что лежит за ее пределами. Ограничивая субъект, SADT-модель помогает сконцентрировать внимание именно на описываемой системе и позволяет избежать включения

посторонних субъектов. Вот почему утверждается, что SADT-модель должна иметь единственный субъект.

С определением модели тесно связана позиция, с которой наблюдается система и создается ее модель. Поскольку качество описания системы резко снижается, если оно не сфокусировано ни на чем, SADT требует, чтобы модель рассматривалась все время с одной и той же позиции. Эта позиция называется "точкой зрения" данной модели. "Точку зрения" лучше всего представлять себе как место (позицию) человека или объекта, в которое надо встать, чтобы увидеть систему в действии. С этой фиксированной точки зрения можно создать согласованное описание системы так, чтобы в модели не смешивались бы несвязанные описания.

После того как определены субъект, цель и точка зрения модели, начинается первая интеграция процесса моделирования по методологии SADT. Субъект определяет, что включить в модель, а что исключить из нее. Точка зрения диктует автору модели выбор нужной информации о субъекте и форму ее подачи. Цель становится критерием окончания моделирования. Конечным результатом этого процесса является набор тщательно взаимоувязанных описаний, начиная с описания самого верхнего уровня всей системы и кончая подробным описанием деталей или операций системы.

Одной из наиболее важных особенностей методологии SADT является постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель.

На рис. 3.2 приведены четыре диаграммы и их взаимосвязи, показана структура SADT-модели. Каждый компонент модели может быть декомпозирован на другой диаграмме. Каждая диаграмма иллюстрирует "внутреннее строение" блока на родительской диаграмме. Дуги, входящие в блок и выходящие из него на диаграмме верхнего уровня, являются точно теми же самыми, что и дуги, входящие в диаграмму нижнего уровня и выходящие из нее, потому что блок и диаграмма представляют одну и ту же часть системы.

Построение SADT-модели начинается с представления всей системы в виде простейшей компоненты - одного блока и дуг, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок представляет всю систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг – они также представляют полный набор внешних интерфейсов системы в целом.

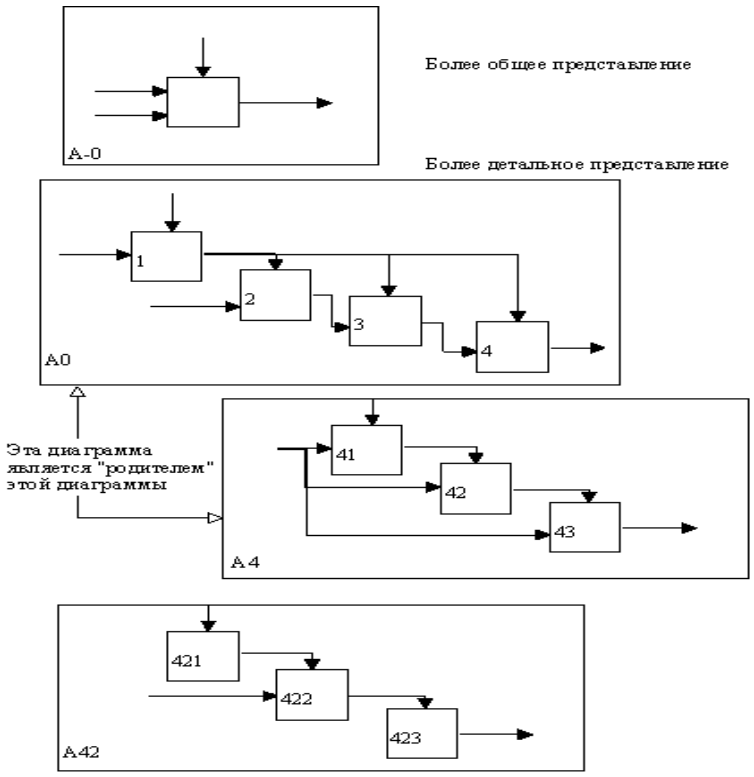

Рис. 3.2 – Структура SADT-модели. Декомпозиция диаграмм

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки представляют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых представлена как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом для более детального представления. Во всех случаях

каждая подфункция может содержать только те элементы, которые входят в исходную функцию.

Модель SADT представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, которые представлены в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах. Каждая детальная диаграмма является декомпозицией блока из более общей диаграммы. На каждом шаге декомпозиции более общая диаграмма называется *родительской* для более детальной диаграммы.

Некоторые дуги присоединены к блокам диаграммы обоими концами, у других же один конец остается не присоединенным. Не присоединенные дуги соответствуют входам, управлениям и выходам родительского блока. Источник или получатель этих пограничных дуг может быть обнаружен только на родительской диаграмме.

Каждый блок на диаграмме имеет свой номер. Блок любой диаграммы может быть далее описан диаграммой нижнего уровня, которая, в свою очередь, может быть далее детализирована с помощью необходимого числа диаграмм. Таким образом, формируется иерархия диаграмм.

Для того, чтобы указать положение любой диаграммы или блока в иерархии, используются номера диаграмм. Например, A21 является диаграммой, которая детализирует блок 1 на диаграмме A2. Аналогично, A2 детализирует блок 2 на диаграмме A0. Номером самого верхнего блока является A-0.

Блоки SADT никогда не размещаются на диаграмме случайным образом. Они размещаются по степени важности, как ее понимает автор диаграммы. В SADT этот относительный порядок называется доминированием. Доминирование понимается как влияние, которое один блок оказывает на другие блоки диаграммы. Например, самым доминирующим блоком диаграммы может быть либо первый из требуемой последовательности функций, либо планирующая или контролирующая функция, влияющая на все другие функции. Наиболее доминирующий блок обычно размещается в верхнем левом углу диаграммы, а

наименее доминирующий - в правом нижнем углу. В результате получается "ступенчатая" схема, подобная представленной на рисунке 3.3. Расположение блоков на странице отражает авторское определение доминирования. Таким образом, топология диаграммы показывает, какие функции оказывают большее влияние на остальные. Чтобы подчеркнуть это, SADT-аналитик может перенумеровать блоки в соответствии с порядком их доминирования.

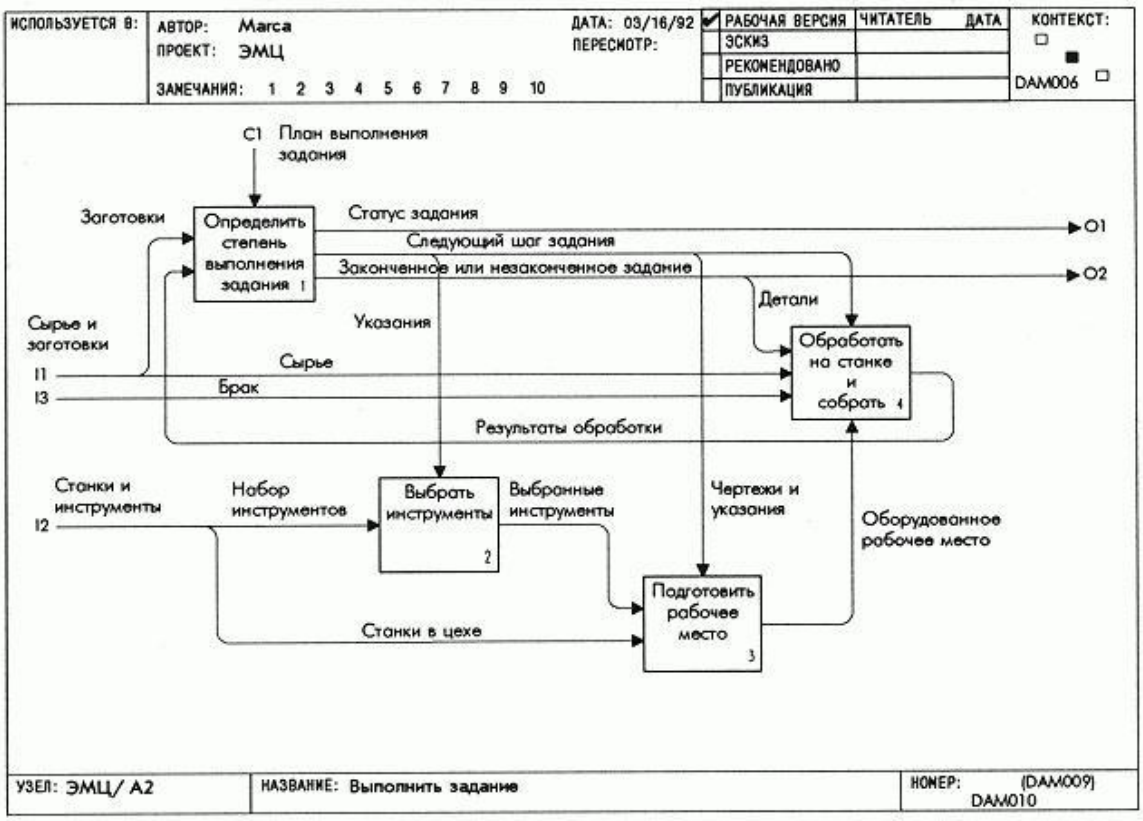

Рис. 3.3 – Пример SADT-диаграммы.

В методологии SADT требуется только пять типов взаимосвязей между блоками для описания их отношений: управление, вход, обратная связь по управлению, обратная связь по входу, выход-механизм. Связи по управлению и входу являются простейшими, поскольку они отражают прямые воздействия, которые интуитивно понятны и очень просты.

Отношение управления возникает тогда, когда выход одного блока непосредственно влияет на блок с меньшим доминированием. Например, блок A1 влияет на блоки A2 и A3 (см. рис. 3.3). Отношение входа возникает тогда, когда

выход одного блока становится входом для блока с меньшим доминированием, например, выход блока А2 становится входом функции А3.

Обратная связь по управлению и обратная связь по входу являются более сложными, поскольку они представляют итерацию или рекурсию, т.е. выходы из одной функции влияют на будущее выполнение других функций, что впоследствии влияет на исходную функцию. Обратная связь по управлению возникает тогда, когда выход некоторого блока влияет на блок с большим доминированием. Связь по входной обратной связи имеет место тогда, когда выход одного блока становится входом другого блока с большим доминированием.

Связи "выход-механизм" встречаются нечасто и представляют особый интерес. Они отражают ситуацию, при которой выход одной функции становится средством достижения цели для другой. Например, на рис. 3.3 представлена функция 3 «Подготовить рабочее место», имеющая выход «оборудованное рабочее место», который, в свою очередь, является механизмом для блока 4. Это означает, что оборудованное рабочее место необходимо для того, чтобы начать процесс обработки. В этом случае дуга механизма обозначает строго последовательную взаимосвязь: приготовления должны быть завершены до начала работы. Поэтому связи "выход-механизм" характерны при распределении источников ресурсов (например, требуемые инструменты, обученный персонал, физическое пространство, оборудование, финансирование, материалы).

Разветвления дуг, изображаемые в виде расходящихся линий, означают, что все содержимое дуг или его часть может появиться в каждом ответвлении дуги. Дуга всегда помечается до разветвления, чтобы дать название всему набору. Кроме того, каждая ветвь дуги может быть помечена или не помечена в соответствии со следующими правилами:

– непомеченные ветви содержат все объекты, указанные в метке дуги перед разветвлением (т.е. все объекты принадлежат этим ветвям);

– ветви, помеченные после точки разветвления, содержат все объекты или их часть, указные в метке дуги перед разветвлением (т.е. каждая метка ветви уточняет, что именно содержит ветвь).

Слияние дуг в SADT, изображаемое как сходящиеся вместе линии, указывает, что содержимое каждой ветви идет на формирование метки для дуги, являющейся результатом слияния исходных дуг. После слияния результирующая дуга всегда помечается для указания нового набора объектов, возникшего после объединения. Кроме того, каждая ветвь перед слиянием может помечаться или не помечаться в соответствии со следующими правилами:

– непомеченные ветви содержат все объекты, указанные в общей метке дуги после слияния (т.е. все объекты исходят из всех ветвей);

– помеченные перед слиянием ветви содержат все или некоторые объекты из перечисленных в общей метке после слияния (т.е. метка ветви ясно указывает, что содержит ветвь).

Хорошая методология структурного анализа, позволяющая создавать отдельные диаграммы, должна гарантировать правильное соединение всех диаграмм для образования согласованной модели. SADT-диаграммы имеют внешние дуги – дуги, как бы выходящие наружу и ведущие к краю страницы. Эти дуги являются интерфейсом между диаграммой и остальной частью модели. SADT требует, чтобы все внешние дуги диаграммы были согласованы с дугами, образующими границу этой диаграммы. Другими словами, диаграмма должна быть "состыкована" со своей родительской диаграммой. Обычно это означает, что внешние дуги согласованы по числу и наименованию (но не обязательно по расположению) с дугами, касающимися декомпозированного блока родительской диаграммы.

В SADT принята система обозначений, позволяющая аналитику точно идентифицировать и проверять связи по дугам между диаграммами. Эта схема кодирования дуг –"ICOM" - получила название по первым буквам английских эквивалентов слов вход (Input), управление (Control), выход (Output), механизм (Mechanism). Коды ICOM чрезвычайно эффективны, поскольку они позволяют аналитику быстро проверять согласованность внешних дуг диаграммы с граничными дугами соответствующего блока родительской диаграммы.

Они также обеспечивают согласованность декомпозиции, поскольку все дуги, входящие в диаграмму и выходящие из нее, должны быть учтены (см. рис. 3.3). Одним из способов такой стыковки может служить присваивание кодов ICOM внешним дугам новой диаграммы согласно следующим правилам:

– представьте себе рисунок новой диаграммы внутри разлагаемого блока. Продлите внешние дуги почти до края диаграммы. Зрительно соедините каждую внешнюю дугу диаграммы с соответствующей граничной дугой декомпозируемого блока.

– присвойте код каждой зрительной связи. Используйте I для входных дуг, C - для связей между дугами управления, O - для связей между выходными дугами, M - для связей между дугами механизма.

– добавьте после каждой буквы цифру, соответствующую положению данной дуги среди других дуг того же типа, касающихся родительского блока. Причем входные и выходные дуги пересчитываются сверху вниз, а дуги управлений и механизмов пересчитываются слева направо. Теперь запишите каждый код около окончания каждой внешней дуги (см. рис. 3.3).

Вопросы для самоконтроля:

1. Какие основные составляющие методологии SADT?
2. Что из себя представляет функциональный блок?
3. В чем заключается цель SADT-модели?
4. Что является субъектом моделирования в SADT-модели?
5. Что такое «точка зрения модели»?
6. Как взаимосвязаны диаграммы SADT-модели между собой?
7. Есть ли связь между функциональным блоком и диаграммы?
8. Какие диаграммы называются родительскими?
9. Как нумеруются диаграммы и блоки?
10. Что такое доминирование SADT-модели?
11. Какие типы взаимосвязей между блоками существуют?
12. Как формируются ICOM-коды?

3.3 Моделирование потоков данных (процессов)

В основе данной методологии лежит построение модели анализируемой ИС - проектируемой или реально существующей. В соответствии с методологией модель системы определяется как иерархия диаграмм потоков данных, описывающих процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процесс становятся элементарными и детализировать их далее невозможно.

Источники информации (внешние сущности) порождают информационные потоки, переносящие информацию к подсистемам или процессам. Те, в свою очередь, преобразуют информацию и порождают новые потоки, которые переносят информацию к другим процессам или подсистемам, накопителям данных или внешним сущностям - потребителям информации. Таким образом, основными компонентами диаграмм потоков данных являются: внешние сущности; системы/подсистемы; процессы; накопители данных; потоки данных.

Внешняя сущность представляет собой материальный предмет или физическое лицо, представляющее собой источник или приемник информации, например, заказчики, поставщики, клиенты. Определение некоторого объекта или системы в качестве внешней сущности указывает на то, что она находится за пределами границ анализируемой ИС. В процессе анализа некоторые внешние сущности могут быть перенесены внутрь диаграммы анализируемой ИС, если это необходимо, или, наоборот, часть процессов ИС может быть вынесена за пределы диаграммы и представлена как внешняя сущность.

Внешняя сущность обозначается квадратом (рис. 3.4), расположенным как бы "над" диаграммой и бросающим на нее тень, для того, чтобы можно было выделить этот символ среди других обозначений.

Рис. 3.4 – Внешняя сущность

При построении модели сложной ИС она может быть представлена в самом общем виде на так называемой контекстной диаграмме в виде одной системы как единого целого, либо может быть декомпозирована на ряд подсистем.

Вид подсистемы (или системы) на контекстной диаграмме изображен на рис. 3.5. Номер подсистемы служит для ее идентификации. В поле имени вводится наименование подсистемы в виде предложения с подлежащим и соответствующими определениями и дополнениями.

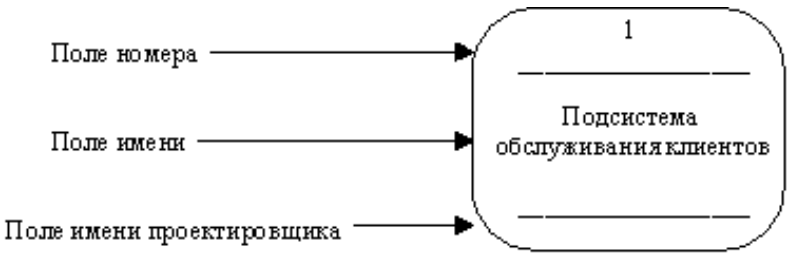

Рис. 3.5 – Подсистема

Процесс представляет собой преобразование входных потоков данных в выходные в соответствии с определенным алгоритмом. Физически процесс может быть реализован различными способами: это может быть подразделение организации, выполняющее обработку входных документов и выпуск отчетов; программа; аппаратно-реализованное логическое устройство и т.д. Процесс на диаграмме потоков данных изображается, как показано на рис. 3.6.

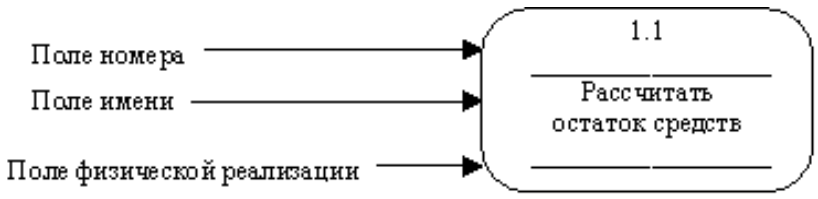

Рис. 3.6 – Процесс

Номер процесса служит для его идентификации. В поле имени вводится наименование процесса в виде предложения с активным недвусмысленным глаголом в неопределенной форме (вычислить, рассчитать, проверить, определить, создать, получить), за которым следуют существительные в винительном падеже, например: "Ввести сведения о клиентах"; "Выдать информацию о текущих расходах"; "Проверить кредитоспособность клиента".

Информация в поле физической реализации показывает, какое подразделение организации, программа или аппаратное устройство выполняет данный процесс.

Накопитель данных представляет собой абстрактное устройство для хранения информации, которую можно в любой момент поместить в накопитель и через некоторое время извлечь, причем способы помещения и извлечения могут быть любыми.

Накопитель данных может быть реализован физически в виде ящика в картотеке, таблицы в оперативной памяти, файла на магнитном носителе и т.д. Накопитель данных на диаграмме потоков данных изображается, как показано на рис. 3.7.

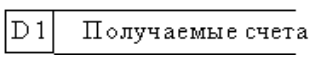

Рис. 3.7 – Накопитель данных

Накопитель данных идентифицируется буквой "D" и произвольным числом. Имя накопителя выбирается из соображения наибольшей информативности для проектировщика. Накопитель данных в общем случае является прооб-

разом будущей базы данных и описание хранящихся в нем данных должно быть увязано с информационной моделью.

Поток данных определяет информацию, передаваемую через некоторое соединение от источника к приемнику. Реальный поток данных может быть информацией, передаваемой по кабелю между двумя устройствами, пересылаемыми по почте письмами, магнитными лентами или дискетами, переносимыми с одного компьютера на другой и т.д.

Поток данных на диаграмме изображается линией, оканчивающейся стрелкой, которая показывает направление потока (рис. 3.8). Каждый поток данных имеет имя, отражающее его содержание.

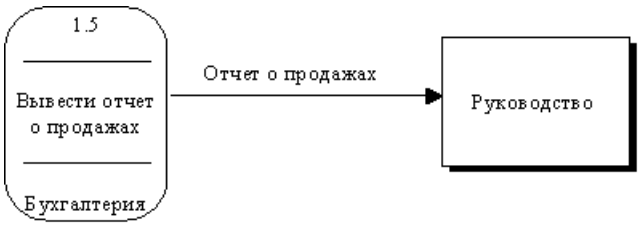

Рис. 3.8 – Поток данных

Первым шагом при построении иерархии диаграмм потоков данных является построение контекстных диаграмм. Обычно при проектировании относительно простых ИС строится единственная контекстная диаграмма в которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы.

Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и кроме того, единственный главный процесс не раскрывает структуры распределенной системы. Признаками сложности (в смысле контекста) могут быть:

– наличие большого количества внешних сущностей;

– распределенная природа системы;

–многофункциональность системы с уже сложившейся или выявленной группировкой функций в отдельные подсистемы.

Для сложных ИС строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не единственный главный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

Иерархия контекстных диаграмм определяет взаимодействие основных функциональных подсистем проектируемой ИС как между собой, так и с внешними входными и выходными потоками данных и внешними объектами (источниками и приемниками информации), с которыми взаимодействует ИС. Разработка контекстных диаграмм решает проблему строгого определения функциональной структуры ИС на самой ранней стадии ее проектирования, что особенно важно для сложных многофункциональных систем, в разработке которых участвуют разные организации и коллективы разработчиков. После построения контекстных диаграмм полученную модель следует проверить на полноту исходных данных об объектах системы и изолированность объектов (отсутствие информационных связей с другими объектами).

Для каждой подсистемы, присутствующей на контекстных диаграммах, выполняется ее детализация при помощи DFD. Каждый процесс, в свою очередь, может быть детализирован при помощи DFD или мини-спецификации. При детализации должны выполняться следующие правила:

– правило балансировки - означает, что при детализации подсистемы или процесса детализирующая диаграмма в качестве внешних источников/приемников данных может иметь только те компоненты (подсистемы, процессы, внешние сущности, накопители данных), с которыми имеет информационную связь детализируемая подсистема или процесс на родительской диаграмме;

– правило нумерации - означает, что при детализации процессов должна поддерживаться их иерархическая нумерация. Например, процессы, детализирующие процесс с номером 12, получают номера 12.1, 12.2, 12.3 и т.д.

Мини-спецификация должна формулировать его основные функции таким образом, чтобы в дальнейшем специалист, выполняющий реализацию проекта, смог выполнить их или разработать соответствующую программу. Мини-спецификация является конечной вершиной иерархии DFD. Решение о завершении детализации процесса и использовании мини-спецификации принимается исходя из следующих критериев:

– наличия у процесса относительно небольшого количества входных и выходных потоков данных (2-3 потока);

– возможности описания преобразования данных процессом в виде известного последовательного алгоритма;

– выполнения процессом единственной логической функции преобразования входной информации в выходную;

– возможности описания логики процесса при помощи мини-спецификации небольшого объема (не более 20-30 строк).

После построения законченной модели системы ее необходимо верифицировать (проверить на полноту и согласованность). В полной модели все ее объекты (подсистемы, процессы, потоки данных) должны быть подробно описаны и детализированы. Выявленные не детализированные объекты следует детализировать, вернувшись на предыдущие шаги разработки.

В согласованной модели для всех потоков данных и накопителей данных должно выполняться правило сохранения информации: все поступающие куда-либо данные должны быть считаны, а все считываемые данные должны быть записаны.

Вопросы для самоконтроля:

1. Что является основными компонентами диаграмм потоков данных?

2. Что собой представляет внешняя сущность на диаграммах потоков данных?

3. Как отображаются подсистемы и процессы на диаграмме потоков данных?

4. Какой смысл ввода накопителя данных на диаграммах потоков данных?

5. Как отображается поток данных на диаграмме потоков данных?

6. Какие существуют критерии окончания процесса декомпозиции на диаграммах потоков данных?

4 Объектно-ориентированный подход к проектированию информационных систем

4.1 Общие сведения об объектно-ориентированном проектировании информационных систем

Объектно-ориентированное проектирование — это подход к решению задач с использованием моделей, основанных на понятиях реального мира [7]. Фундаментальным элементом является объект, объединяющий структуру данных с поведением. Характеристики объектно-ориентированного подхода включают в себя индивидуальность, классификацию, наследование и полиморфизм.

Индивидуальность означает, что данные делятся на дискретные сущности, хорошо отличимые друг от друга. Эти сущности называются объектами. Объекты могут быть конкретными, как, например, файл в составе файловой системы, или концептуальными, как политика планирования в многопроцессорной операционной системе. Каждый объект обладает своей собственной внутренней индивидуальностью. Два объекта различимы даже в том случае, если значения всех их атрибутов (таких как имя или размер) одинаковы.

Классификация означает, что объекты с одинаковыми структурами данных (атрибутами) и поведением (операциями) группируются в классы. Класс — это абстракция, описывающая свойства, важные для конкретного приложения, и игнорирующая все остальное. Любой выбор классов произволен и зависит от приложения. Каждый класс описывает множество индивидуальных объектов, которое может быть бесконечным. Каждый объект из этого множества называется экземпляром класса. Объект имеет свои собственные значения атрибутов, но названия атрибутов и операций являются общими для всех экземпляров класса.

Наследование — это наличие у разных классов, образующих иерархию, общих атрибутов и операций. Суперкласс задает наиболее общую информацию, которую затем уточняют и улучшают его подклассы. Каждый подкласс соеди-

няет в себе, то есть наследует, все черты его суперкласса, к которым добавляет собственные уникальные черты. Подклассам необязательно воспроизводить все черты суперкласса. Возможность выделять общие черты нескольких классов в суперкласс значительно сокращает количество повторений в проектах и программах и является одним из основных достоинств объектно-ориентированной технологии.

Полиморфизм означает, что одна и та же операция может подразумевать разное поведение в разных классах. Операция — это процедура или трансформация, которую объект выполняет сам или которая осуществляется с данным объектом. Реализация операций в конкретном классе называется методом. Поскольку объектно-ориентированная операция является полиморфной, в разных классах объектов она может быть реализована разными методами.

В реальном мире операция является абстрагированием похожего поведения у объектов одного рода. Каждый объект «сам знает», как выполнить свои собственные операции. В объектно-ориентированных языках программирования выбор подходящего метода для реализации операции осуществляется автоматически, исходя из имени операции и класса объекта, к которому она относится. Клиенту операции нет необходимости знать о том, сколько еще методов могут реализовывать данную полиморфическую операцию. Разработчики могут добавлять новые классы, не меняя существующий код, при условии, что они предоставляют методы для каждой возможной операции.

Объектно-ориентированное проектирование — это концептуальный процесс, независимый от языка программирования, по крайней мере, до последних этапов. Фактически это образ мышления, а не методика программирования. Главное преимущество объектно-ориентированного проектирования состоит в том, что оно помогает тем, кто пишет спецификации, разработчикам и заказчикам ясно выражать абстрактные концепции и обсуждать их друг с другом. Таким образом, облегчается составление спецификаций, анализ, документирование и определение интерфейсов и, конечно же, программирование.

Объектно-ориентированный подход состоит из построения модели приложения и последующей ее детализации. Одна и та же система обозначений используется на всем протяжении процесса разработки, начиная с анализа и заканчивая реализацией. Информация, добавленная на одном из этапов, не будет утеряна или даже преобразована при переходе к следующему этапу. Описываемая методология включается в себя следующие этапы:

1) **Концептуализация системы.** Разработка программного обеспечения начинается с бизнес-аналитиков или пользователей, которые придумывают приложение и формируют первичные требования к нему.

2) **Анализ.** Аналитик тщательно исследует и переформулирует требования, конструируя модели, исходя из концепций системы. Аналитик должен работать с заказчиком, чтобы добиться понимания задачи, потому что формулировки редко оказываются полными и корректными. Аналитическая модель — это сжатая и точная абстракция того, что именно должна сделать система (а не то, каким образом это будет сделано). Аналитическая модель не должна содержать никаких решений относительно реализации. Аналитическая модель состоит из двух частей: модели предметной области — описания объектов реального мира, отражаемых системой, и модели приложения — описания видимых пользователю частей самого приложения. Например, для приложения биржевого маклера объектами предметной области могут быть акции, облигации, торги и комиссия. Объекты модели приложения могут управлять выполнением торгов и отображать результаты. Хорошая модель должна быть доступной для понимания и критики со стороны экспертов, не являющихся программистами.

3) **Проектирование системы.** Команда разработчиков продумывает стратегию решения задачи на высшем уровне, определяя архитектуру системы. На этом этапе определяются политики, которые послужат основой для принятия решений на следующих этапах. Проектировщик системы должен выбрать параметры системы, по которым будет проводиться оптимизация, предложить стратегический подход к задаче, провести предварительное распределение ресурсов. Например, проектировщик может решить, что любые изменения изоб-

ражения на экране рабочей станции должны быть быстрыми и плавными, даже при перемещении и закрытии окон. На основании этого решения он может выбрать подходящий протокол обмена и стратегию буферизации памяти.

4) **Проектирование классов.** Проектировщик классов уточняет аналитическую модель в соответствии со стратегией проектирования системы. Он прорабатывает объекты предметной области и объекты модели приложения, используя одинаковые объектно-ориентированные концепции и обозначения, несмотря на то, что эти объекты лежат в разных концептуальных плоскостях. Цель проектирования классов состоит в том, чтобы определить, какие структуры данных и алгоритмы требуются для реализации каждого класса.

5) **Реализация.** Ответственные за реализацию занимаются переводом классов и отношений, образовавшихся на предыдущем этапе, на конкретный язык программирования, воплощением их в базе данных или аппаратном обеспечении. Никаких усложнений на этом этапе быть не должно, потому что все ответственные решения уже были приняты на предыдущих этапах. В процессе реализации необходимо использовать технологии разработки программного обеспечения, чтобы соответствие кода проекту было очевидным, а система оставалась гибкой и расширяемой.

Объектно-ориентированные концепции действуют на протяжении всего жизненного цикла информационной системы. Одни и те же классы будут переходить от одного этапа к другому без всяких изменений в нотации, хотя на последних этапах они существенно обрастут деталями. Концепции индивидуальности, классификации, полиморфизма и наследования действуют на протяжении всего процесса разработки.

Некоторые классы не входят в аналитическую модель. Они появляются позднее, на этапах проектирования и реализации. Например, структуры данных, подобные деревьям, хэш-таблицам и связным спискам, редко появляются в реальном мире и обычно невидимы пользователям. Проектировщики добавляют их в систему для того, чтобы обеспечить поддержку выбранных алгоритмов.

Объекты структур данных существуют внутри компьютера и не являются непосредственно наблюдаемыми.

Тестирование не рассматривается как отдельный этап. Тестирование очень важно, но оно должно быть частью системы контроля качества, которая применяется на протяжении всего жизненного цикла. Разработчики должны сравнивать аналитические модели с реальностью. Они должны проверять проектировочные модели на наличие ошибок различных видов, а не только тестировать корректность реализации. Выделение всех операций по контролю качества в отдельный этап стоит дороже и оказывается менее эффективно.

В 1996 году группа управления объектами (Object Management Group — OMG) объявила конкурс на лучший стандарт обозначений для объектно-ориентированного моделирования. В этом конкурсе приняло участие несколько компаний. В результате их предложения были объединены в конечную систему. В ноябре 1997 года группа OMG приняла получившийся в результате унифицированный язык моделирования (UML — Unified Modeling Language) в качестве стандарта. Компании, принимавшие участие в конкурсе, передали права на UML группе OMG, которая стала владельцем торговой марки и спецификаций UML. Эта группа управляет дальнейшим совершенствованием языка UML. Система обозначений UML оказалась настолько удачной, что вытеснила практически все другие системы. В настоящий момент действует вторая версия этого языка [9].

Вопросы для самоконтроля:

1. Что является фундаментальным элементом в объектно-ориентированном проектировании?

2. Что означает индивидуальность в объектно-ориентированном проектировании?

3. Что означает классификация в объектно-ориентированном проектировании?

4. Что означает наследование в объектно-ориентированном проектировании?

5. Что означает полиморфизм в объектно-ориентированном проектировании?

6. Из каких этапов состоит объектно-ориентированный подход?

7. Какой синтаксис сейчас используется для отображения объектно-ориентированных моделей проектирования?

4.2 Объектно-ориентированные концепции

В объектно-ориентированной технологии широко используются несколько базовых концепций. Они не ограничиваются рамками объектно-ориентированных систем, но объектная ориентированность означает, прежде всего, поддержку этих концепций.

Абстракция. Абстракция означает сосредоточение на важнейших аспектах приложения и игнорирование всех остальных. Сначала принимается решение о том, что представляет собой объект и что он делает, а затем подбирается способ его реализации. Использование абстракций позволяет сохранить свободу принятия решений как можно дольше благодаря тому, что детали не фиксируются раньше времени. Большинство современных языков программирования позволяют абстрагировать данные, на наследование и полиморфизм значительно расширяют возможности концепции абстрагирования. Умение создавать абстракции, вероятно, является самым важным качеством для объектно-ориентированного разработчика.

Инкапсуляция. Инкапсуляция, или, иначе говоря, сокрытие информации, состоит в отделении внешних аспектов объекта, доступных другим объектам, от деталей внутренней реализации, которые от других объектов скрываются. Инкапсуляция исключает возникновение взаимозависимости участков программы, из-за которых небольшие изменения приводят к значительным непредвиденным последствиям. Реализация объекта может быть изменена безо всяких

последствий для использующих его приложений. Изменение реализации может быть предпринято для повышения производительности, устранения ошибки, консолидации кода или для подготовки к переносу программы на другие системы.

Объединение данных и поведения. При вызове операции не нужно беспокоиться о том, сколько реализаций этой операции существует в системе. Полиморфизм операторов перекладывает ответственность за выбор подходящей реализации с вызывающего кода на иерархию классов. Рассмотрим в качестве примера обычный (не объектно-ориентированный) код, отображающий содержимое окна. Такой код должен учитывать тип каждой фигуры (многоугольник, окружность, текст) и вызывать соответствующие типу процедуры. Объектно-ориентированный код будет вызывать для каждой фигуры метод «прорисовать». Каждый объект выберет подходящую процедуру согласно своему классу. Это облегчает поддержку программы, потому что добавление нового класса не требует изменения вызывающего кода. В объектно-ориентированной системе иерархия структур данных соответствует иерархии наследования операций (рис. 4.1).

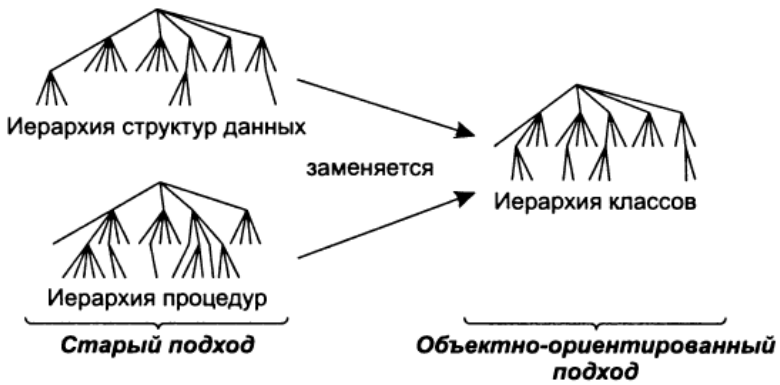

Рис. 4.1 – Единая иерархия операций и структур данных

Совместное использование. Объектно-ориентированные технологии способствуют совместному использованию сущностей на самых разных уровнях. Наследование структур данных вместе с поведением дает возможность подклассам совместно использовать общий код. Совместное использование при

наследовании является одним из главных преимуществ объектно-ориентированных языков. Однако, концептуальная ясность, проистекающая из осознания того, что разные операции на деле представляют собой одно и то же, оказывается важнее, чем экономия кода. Благодаря этому сокращается количество различных ситуаций, которые требуется понять и проанализировать. Объектно-ориентированная разработка позволяет не только совместно работать с общей информацией внутри приложения, но и повторно использовать проекты и код в последующих программах. Объектно-ориентированные средства помогают строить библиотеки повторно используемых компонентов.

Выделение сущности объекта. Объектно-ориентированная технология выделяет то, чем объект *является*, а не то, как он *используется*. Использование объекта зависит от особенностей приложения и часто изменяется в процессе разработки. По мере уточнения требований, черты объекта остаются более стабильными, чем детали его использования, поэтому системы, основанные на объектной структуре, в конечном счете, оказываются более стабильными. При объектно-ориентированной разработке большее внимание уделяется структурам данным и меньшее — процедурам, нежели в методологиях, связанных с функциональным разбиением. В этом отношении объектно-ориентированная разработка подобна технологиям информационного моделирования, используемым при проектировании баз данных (за исключением добавленной в объектно-ориентированном подходе концепции поведения, зависящего от класса).

Когда целое больше суммы частей. Все объектно-ориентированные языки характеризуются поддержкой концепций индивидуальности, классификации, полиморфизма и наследования. Каждая из этих концепций может использоваться сама по себе, однако вместе они образуют нечто большее. Преимуществ объектно-ориентированного подхода оказывается больше, чем может показаться с первого взгляда. Выделение основных свойств объекта заставляет разработчика более внимательно относиться к тому, чем объект является и что он делает. В результате система оказывается более ясной, универсальной и

устойчивой, чем в том случае, если бы основное значение придавалось данным и операциям.

Для описания системы с различных точек зрения используют три типа моделей. Модель классов описывает объекты, входящие в состав системы, и отношения между ними. Модель состояний описывает историю жизни объектов. Модель взаимодействий описывает взаимодействия между объектами. Каждая модель применяется на всех этапах проектирования и постепенно обрастает деталями. Полное описание системы требует наличия всех трех моделей.

Модель классов описывает статическую структуру объектов системы и их отношения. Эта модель определяет контекст разработки программы, то есть предметную область. Модель классов изображается на диаграммах классов. Диаграмма классов — это граф, вершинами которого являются классы, а ребрами — их отношения.

Модель состояний описывает изменяющиеся со временем аспекты объектов. Эта модель реализуется посредством диаграмм состояний. Диаграмма состояний — это граф, вершинами которого являются состояния, а ребрами — переходы между состояниями, инициируемые событиями.

Модель взаимодействия описывает кооперацию объектов системы для достижения лучших результатов. Построенные модели начинаются с вариантов использования, которые затем уточняются на диаграммах последовательности и диаграммах деятельности. Вариант использования описывает функциональность системы, то есть то, что системы делает для пользователей. Диаграмма последовательности изображает взаимодействие объектов и временную последовательность этого взаимодействия. Диаграмма деятельности уточняет важные этапы обработки.

Три описанные модели являются связанными между собой составляющими полного описания системы. Центральной является модель классов, поскольку сначала нужно определить, *что* именно изменяется или трансформируется, а затем уже описывать *когда* и *как* это происходит.

Вопросы для самоконтроля:

1. Что означает концепция абстракции в объектно-ориентированном проектировании?

2. Что означает концепция инкапсуляции в объектно-ориентированном проектировании?

3. Что означает концепция объединения данных и поведения в объектно-ориентированном проектировании?

4. Что означает концепция совместного использования в объектно-ориентированном проектировании?

5. Что означает концепция выделения сущности объекта в объектно-ориентированном проектировании?

6. Что означает концепция «когда целое больше суммы частей» в объектно-ориентированном проектировании?

7. Какие типы моделей используются для полного описания систем в объектно-ориентированном проектировании?

4.3 Моделирование классов

4.3.1 Концепции объекта и класса

Цель моделирования классов состоит в описании объектов. Объект — это концепция, абстракция или сущность, обладающая индивидуальностью и имеющая смысл в рамках приложения. Объекты часто бывают имена собственными или конкретными ссылками, которые используются в описании задач или при общении с пользователями. Некоторые объекты существуют или существовали в реальном мире (например, «Джо Смит»*)*, тогда как другие являются сугубо концептуальными сущностями (например, «тестовый прогон № 1234» или «формула корней квадратного уравнения»). Объекты третьего типа (например, «бинарное дерево» и «массив, связанный с переменной а»*)* добавляются в модель в процессе реализации и не имеют никакого отношения к физической ре-

альности. Выбор объектов зависит от природы задачи и от предпочтений разработчика. Корректное решение в данном случае не является единственным.

Все объекты обладают индивидуальностью и потому отличаются друг от друга. Два яблока одинакового цвета, формы и текстуры все равно являются разными яблоками. Вы можете съесть сначала одно из них, а потом второе. Похожие друг на друга близнецы тоже являются независимыми индивидуальностями. Индивидуальность означает, что объекты отличаются друг от друга внутренне, а не по внешним свойствам.

Объект является экземпляром класса. Класс описывает группу объектов с одинаковыми свойствами (атрибутами), одинаковым поведением (операциями), типами отношений и семантикой. В качестве примеров классов можно привести следующие: «человек», «компания», «процесс» и «окно». Каждый человек имеет имя и дату рождения, а также может где-либо работать. Каждый человек-процесс имеет владельца, приоритет и список необходимых ресурсов. Классы часто бывают именами нарицательными и именными группами, которые используются в описании задач или при общении с пользователями.

Объекты одного класса имеют одинаковые атрибуты и формы поведения. Большинство объектов отличаются друг от друга значениями своих атрибутов и отношениями с другими объектами. Однако возможно существование разных объектов с одинаковыми значениями атрибутов, находящихся в одинаковых отношениях со всеми остальными объектами. Выбор классов зависит от природы и области применения приложения и зачастую является субъективным.

Объекты класса имеют общее семантическое значение, помимо обязательных общих атрибутов и поведения. Например, и амбар, и лошадь могут характеризоваться стоимостью и возрастом. С финансовой точки зрения они могли бы относиться к одному классу. Однако если разработчик учтет, что человек может красить амбар и кормить лошадь, то эти две сущности будут отнесены к разным классам. Интерпретация семантики зависит от назначения конкретного приложения и является делом субъективным.

Каждый объект «знает» свой собственный класс. Большинство объектно-ориентированных языков программирования позволяют определять класс объекта во время выполнения программы. Класс объекта — это его неявное свойство.

Если предметом моделирования являются объекты, то почему вообще заходит речь о классах? Все дело в необходимости абстрагирования. Группируя объекты в классы, мы производим абстрагирование в рамках задачи. Именно благодаря абстракциям моделирование оказывается столь мощным инструментом, позволяющим проводить обобщения от нескольких конкретных случаев к множеству подобных альтернатив. Общие определения (такие как название класса и названия атрибутов) хранятся отдельно для каждого класса, а не для каждого экземпляра.

Операции могут быть написаны один раз для целого класса, благодаря чему все объекты класса получают возможность повторно использовать написанный код. Например, все эллипсы имеют общие процедуры прорисовки, вычисления площади и проверки на наличие пересечения с некоторой прямой. У многоугольников будет свой набор процедур. Даже в особых случаях (например, для окружностей и квадратов) могут использоваться универсальные процедуры, хотя специальные алгоритмы могут быть и эффективнее.

Описания понятий объекта и класса не дает достаточной четкости и непригодны для описания сложных приложений. Необходимо средство описания моделей, которое было бы согласованным, точным и простым в использовании. Модели структуры бывают двух типов: диаграммы классов и диаграммы объектов.

Диаграммы классов позволяют описать модель классов и их отношений (а значит и возможные объекты) при помощи графической системы обозначений. Диаграмма классов описывает бесконечное множество диаграмм объектов.

На рис. 4.2 показан класс (слева) и его экземпляры (справа). Объекты *JoeSmith* (ДжоСмит), *MarySharp* (МэриШарп) и безымянная личность являются экземплярами класса *Person* (Человек). В языке UML для обозначения объекта

используется прямоугольник, внутри которого ставится имя объекта, двоеточие и имя класса, к которому относится этот объект. Имя объекта и имя класса подчеркиваются. Также используется дополнительное соглашение, которое состоит в том, что имена объектов и классов выделяется полужирным шрифтом.

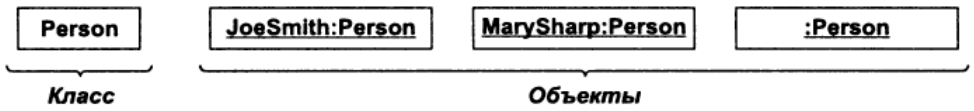

Рис 4.2 – Отображение объектов и классов в нотации UML

Для обозначения класса в UML тоже используется прямоугольник. Имя класса указывается полужирным шрифтом, располагаясь посередине прямоугольника, и начинается с заглавной буквы. Эти соглашения используются для ссылок на объекты, классы и другие конструкции. Соглашение об использовании заглавных букв между строчных популярно в литературе, посвященной объектно-ориентированным технологиям, но не является требованием языка UML.

Значение — это элемент данных. Значения можно определить, изучив примеры, приведенные в документации по поставленной задаче. Атрибут — это именованное свойство класса, описывающее значение, которое может иметь каждый объект класса. Атрибуты — это прилагательные. Они получаются абстрагированием типичных значений. Можно привести следующую аналогию: объект относится к классу так, как значение относится к атрибуту. В моделях классов преобладают структурные конструкции. Атрибуты имеют не столь важное значение и служат для уточнения характеристик классов и отношений.

Атрибутами объектов класса *Человек* являются *имя*, *датаРождения* и *вес*. У каждого конкретного объекта атрибут принимает свое конкретное значение. Например, у объекта *ДжонСмит* атрибут *датаРождения* может иметь значение «21 октября 1983». У разных объектов один и тот же атрибут может иметь как разные, так и одинаковые значения. Имя атрибута уникально в рамках клас-

са (но не обязательно уникально во множестве всех классов), поэтому у разных классов может быть атрибут с одним и тем же названием.

Не следует путать значения с объектами. Атрибут должен описывать значения, а не объекты. В отличие от объектов значения не обладают индивидуальностью. Например, все экземпляры числа 17 неразличимы между собой. Точно так же неразличимы экземпляры строкового значения «Канада». С другой стороны, страна, которая называется Канада, является объектом, у которого атрибут *название* имеет значение «Канада».

На рис. 4.3 показана система обозначений атрибутов. Класс *Person* (*Человек*) имеет атрибуты *name* (*имя*) и *birthdate* (*датарождения*). *Name* — это строка, а *birthdate* — дата. У одного из объектов класса *Person* атрибут *name* имеет значение «Джо Смит», а *birthdate* имеет значение «21 октября 1983». У другого объекта того же класса атрибут *name* имеет значение «Мэри Шарп», а атрибут *birthdate* имеет значение «16 марта 1950».

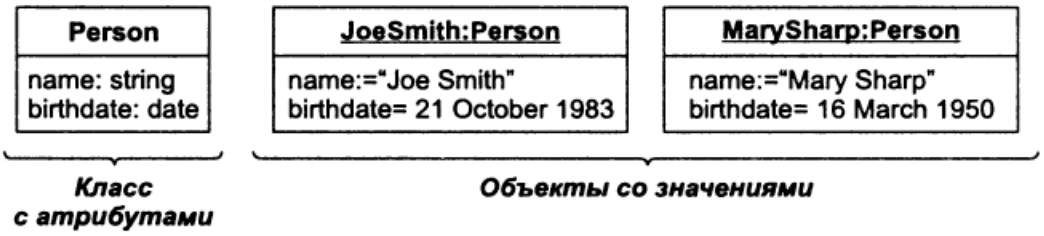

Рис. 4.3 – Отображение атрибутов в нотации UML

Согласно системе обозначений UML, атрибуты указываются во втором (сверху) отделе прямоугольника, обозначающего класс. После каждого атрибута могут быть указаны необязательные сведения о нем (например, тип и значение по умолчанию). Перед значением по умолчанию ставится знак равенства. Название атрибута указывается обычным шрифтом (не полужирным) и первая буква не делается заглавной.

Иногда среда реализации может требовать наличия у объекта уникального идентификатора. Идентификаторы всегда неявным образом присутствуют в модели классов. Их не следует указывать явно. Большинство объектно-

ориентированных языков генерируют идентификаторы для ссылок на объекты автоматически. Так же легко определить идентификаторы и для баз данных. Идентификаторы являются артефактом вычислительной системы и не имеют внутреннего смысла.

Не следует путать внутренние идентификаторы с атрибутами. Внутренние идентификаторы вводятся исключительно для удобства реализации и не имеют никакого смысла в контексте приложения. Такие атрибуты, как индивидуальный номер налогоплательщика, серийный номер и телефонный номер, не являются внутренними идентификаторами, поскольку они имеют значение в реальном мире. Их можно назвать полноправными атрибутами.

Операция — это функция или процедура, которая может быть применена к объектам класса. Операциями класса *Компания* могут быть *нанять*, *уволить*, *выплатитьДивиденды*. Все объекты одного класса имеют общий список операций. Каждая операция в качестве неявного аргумента принимает свой целевой объект. Поведение операции зависит от класса целевого объекта. Объект всегда знает свой собственный класс, а потому он всегда знает и правильную реализацию операции.

Одна и та же операция может быть применена к разным классам. Такая операция называется полиморфной: в разных классах она может принимать разные формы. Методом называется реализация операции в конкретном классе. например, класс *Файл* может иметь операцию *печать*. Печать ASCII-файлов, двоичных файлов и цифровых изображений может осуществляется разными методами. Все эти методы с логической точки зрения выполняют одно и то же действие: печать файла. Поэтому они и являются реализациями одной и той же операции *печать*. При этом каждый метод может быть реализован своим собственным кодом.

У операции могут быть и другие аргументы, кроме целевого объекта. Эти аргументы могут быть как значениями, так и другими объектами. Выбор метода зависит только от класса целевого объекта, но не от классов аргументов, которые являются объектами, сколько бы их ни было. В некоторых объектно-

ориентированных языках выбор метода может определяться произвольным числом аргументов, но такая универсальность значительно усложняет семантику модели.

Если операция реализована несколькими методами в разных классах, очень важно, чтобы у всех методов была одна и та же сигнатура — количество и типы аргументов, а также тип возвращаемого значения. Поведение всех методов операции должно быть согласованно. Лучше всего избегать использования одинаковых названий для операций, отличающихся друг от друга с семантической точки зрения, даже если они применяются к разным множествам классов. Например, неправильно было бы использовать название «инверсия» для операций инвертирования матрицы и переворота геометрической фигуры. В большом проекте может потребоваться использование пространств имен для предотвращения возможных проблем.

На рис. 4.4 изображен класс *Person* (*Человек*) с атрибутами *name* (*имя*) и *birthdate* (*датаРождения*) и операциями *changeJob* (*сменитьРаботу*) и *changeAddress* (*сменитьАдрес*). Атрибуты и операции называются составляющими класса. У класса *File* (*Файл*) есть операция *print* (*печать*). У класса *GeometricObject* (*ГеометрическийОбъект*) есть операция *move* (*перемещение*), *select* (*выделение*) и *rotate* (*поворот*). Операция *move* имеет один аргумент *delta* (*смещение*) типа *Vector* (*Вектор*), операция *select* имеет аргумент *p* типа *Point* (*Точка*) и возвращает значение типа *Boolean* (*Логический*). Операция *rotate* имеет аргумент *angle* (*угол*), который относится к типу чисел с плавающей точкой и имеет значение по умолчанию 0.0.

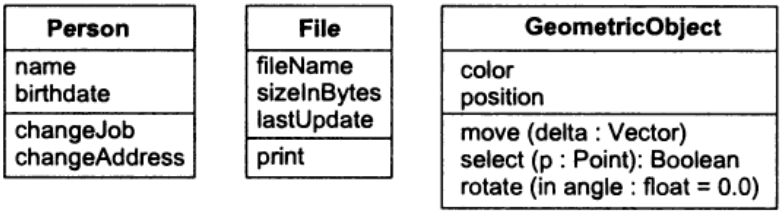

Рис. 4.4 – Отображение операций в нотации UML

Система обозначение UML предписывает перечислять операции в третьем отделе прямоугольника класса. Название операции мы указываем обычным шрифтом и выравниваем по левому краю. Первую букву названия мы не делаем заглавной. После названия операции могут быть указаны необязательные дополнительные сведения, такие как список аргументов и тип возвращаемого результата. Список аргументов указывается в круглых скобках. Аргументы отделяются друг от друга запятыми. Перед типом возвращаемого результата ставится двоеточие. пустой список аргументов в круглых скобках явно показывает, что данная функция не принимает никаких аргументов. Если же списка просто нет, никаких выводов делать нельзя. Мы не указываем операции для отдельных объектов, поскольку у всех объектов одного класса операции одинаковые.

Полная система обозначений классов показана на рис. 4.5. Класс обозначается прямоугольником, который может иметь до трех отделов. Отделы нумеруются сверху вниз. В первом отделе указывается имя класса, во втором — список атрибутов, в третьем — список операций. После названия атрибута может быть указан его тип и значение по умолчанию. После названия операции может быть указан список аргументов и тип возвращаемого значения.

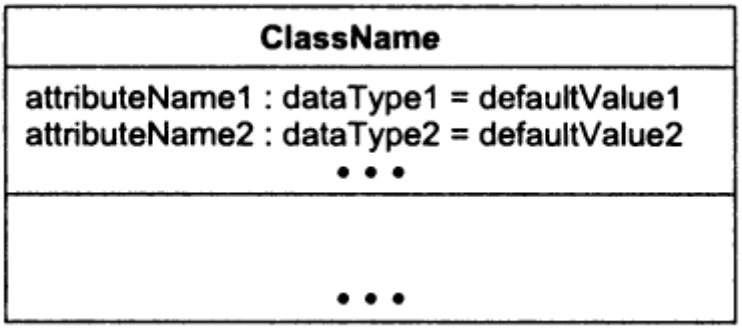

Рис. 4.5 – Общее отображение классов в нотации UML

Каждый аргумент может быть охарактеризован направлением, названием, типом и значением по умолчанию. По направлению аргумент может быть входным (in), выходным (out) или изменяемым (inout). Перед типом аргумента ставится двоеточие. Перед значением по умолчанию ставится знак равенства.

Это значение используется в том случае, если при вызове операции значение аргумента не было указано явно.

Отделы со списком атрибутов и операций являются необязательными элементами системы обозначений. Отсутствие отдела атрибутов говорит о том, что в данном представлении атрибуты не указаны. Отсутствие отдела операций также говорит о том, что в данном представлении не указаны операции. Напротив, наличие пустого отдела говорит о том, что атрибуты или операции отсутствуют.

4.3.2 Концепции связи и ассоциации

Связь — это физическое или концептуальное соединение между объектами. Например, *Джо Смит работает на компанию Симплекс*. В большинстве случаев связь соединяет ровно два объекта, но бывают связи, соединяющие большее количество объектов. С математической точки зрения связь является кортежем, то есть списком объектов. Связь — это экземпляр ассоциаций.

Ассоциация — это описание группы связей, обладающих общей структурой и общей семантикой. Например, *человек может работать на какую-либо компанию.* Связи, являющиеся экземплярами некоторой ассоциации, соединяют объекты тех классов, которые соединены между собой этой ассоциацией. Ассоциация описывает множество потенциальных связей точно так же, как класс описывает множество потенциальных объектов. Связи и ассоциации обычно присутствуют в постановке задачи в виде глаголов.

На рис. 4.6 приведен пример диаграммы классов (верхняя часть) и диаграмма объектов (нижняя часть).

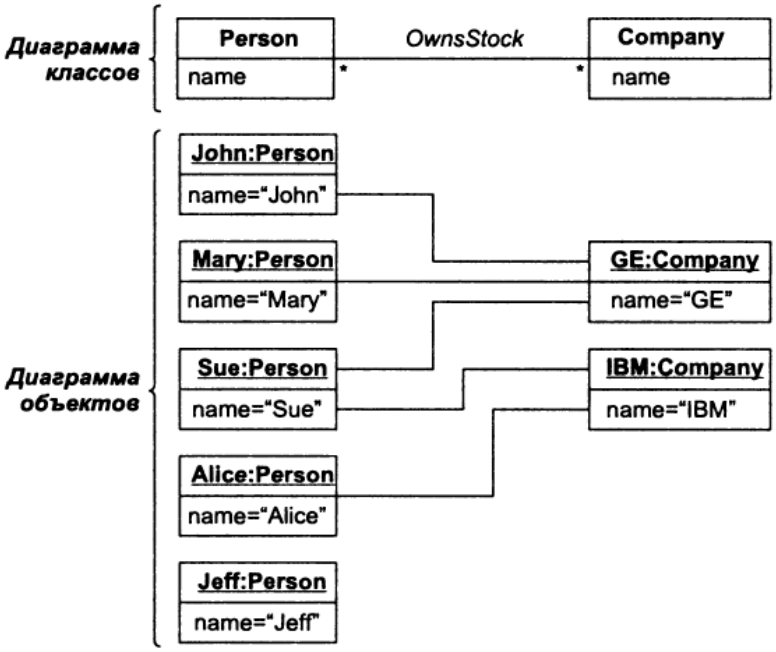

Рис. 4.6 – Отображение ассоциаций и связей в нотации UML

Диаграмма классов показывает, что *человек* может *быть владельцем акций* неопределенного количества *компаний* (от нуля и более). С другой стороны, *владельцами акций* одной *компании* могут *быть* несколько *человек*. На диаграмме объектов показаны примеры. *Джон, Мэри* и *Сью являются владельцами акций General Electric. Сью* и *Элис являются владельцами акций* компании *IBM. Джефф* не купил никаких акций, а потому и связи для него нет. Звездочка на нашем рисунке — это символ кратности. Кратность определяет количество экземпляров одного класса, которые могут быть связаны с одним экземпляром другого класса.

Система обозначений UML предписывает изображать связь как линию между двумя объектами. Линия может состоять из нескольких прямолинейных сегментов. Если у связи есть имя, оно подчеркивается. В нашем примере Джон является владельцем акций General Electric. Ассоциация соединяет между собой классы и тоже обозначается линией (которая тоже может иметь несколько прямолинейных сегментов). В нашем примере люди являются владельцами акций компаний. По нашему соглашению названия связей и ассоциаций выделяются курсивом, а сегменты линий привязываются к прямоугольной сетке. Удобно, по

возможности, расставлять классы так, чтобы ассоциация читалась слева направо.

Название ассоциации указывать не обязательно, если в модели не возникает двусмысленности. Неоднозначность появляется в тех случаях, когда между одними и теми же классами существуют несколько ассоциаций, например, *человек работает на компанию* и *человек является владельцем акций компании.* В этом случае необходимо использовать имена ассоциаций или имена полюсов ассоциаций.

Ассоциации по своей сути являются двусторонними. Название бинарной ассоциации обычно читается в конкретном направлении, но сама ассоциация может быть прослежена в любом направлении. Например, ассоциация *работает на* соединяет *человека* и *компанию*. Обратная ассоциация могла бы называться *оплачивает услуги,* и она соединяла бы *компанию* и *человека.* В реальности оба направления ассоциации имеют одинаково важное значение и относятся к одной и той же ассоциации. Только названия ассоциаций задают им определенные направления.

Разработчики часто реализуют ассоциации в виде ссылок из одного объекта на другой. Ссылка — это атрибут объекта, ссылающийся на другой объект. Например, структура данных класса *Человек* может содержать атрибут *работодатель,* ссылающийся на объект класса *Компания,* а объект класса *Компания* может содержать атрибут *сотрудники,* ссылающийся на множество объектов класса *Человек.* Реализация ассоциаций в виде ссылок вполне приемлема, но моделировать ассоциации таким образом не следует.

Связь — это отношение между объектами. Моделирование связи в виде ссылки скрывает тот факт, что связь не является частью одного из объектов, а зависит от них обоих. Компания не является частью человека, а человек — частью компании. Более того, использование пары ссылок скрывает тот факт, что прямая и обратная ссылки зависят друг от друга. Поэтому все соединения между классами следует моделировать как ассоциации, даже при разработке проектов программ.

В объектно-ориентированном подходе подчеркивается важность инкапсуляции, которая состоит в скрытии деталей реализации внутри класса. Ассоциации особенно важны потому, что они нарушают инкапсуляцию. Ассоциации не могут быть скрыты внутри класса, поскольку они соединяют разные классы. Ассоциации должны считаться равноправными по отношению к классам, в противном случае в программах будут содержаться скрытые предположения и зависимости. Такие программы трудно расширять, а классы, из которых они состоят, сложно использовать повторно.

Кратность — это количество экземпляров одного класса, которые могут быть связаны с одним экземпляром другого класса через одну ассоциацию. Кратность ограничивает количество связанных между собой объектов. Чаще всего рассматриваются два значения кратности: 1 и «много», но общем случае кратность может быть потенциально бесконечным подмножеством неотрицательных целых чисел. На диаграммах UML кратность указывается явно около конца линии, которой обозначается ассоциация. Значение кратности указывается в виде диапазона, например, «1» (равно один), «1..*» (один и более) или «3..5» (от трех до пяти включительно). Специальный символ «*» обозначает слово «много» - нуль и более.

Не следует путать кратность с количеством элементов. Кратность — это ограничение на размер совокупности, а количество элементов — это число элементов, которые фактически входят в совокупность. Следовательно, кратность ограничивает количество элементов.

Кратность зависит от предположений и от определенных разработчиком границ задачи. Нечеткие требования часто затрудняют определение кратности. Не следует сильно беспокоиться о правильных значениях кратности на начальных этапах разработки системы. Сначала следует определить классы и ассоциации, а затем указывать кратность. Если кратность не указана на диаграмме, она считается неопределенной.

Кратность неявно подразумевает наличие полюсов ассоциации. Например, ассоциация один-ко-многим имеет два полюса, у одного из которых указана

кратность «один», а у другого – «много». Концепция полюса ассоциации – одна из важнейших в UML. Полюс ассоциации может иметь не только кратность, но и свое собственное имя.

Имена полюсов ассоциаций часто присутствуют в описаниях задач в виде существительных. Имя полюса указывается около конца ассоциации. На рис. 4.7 *Person* (*Человек*) и *Company* (*Компания*) участвуют в ассоциации *WorksFor* (*РаботаетНа*). Человек по отношению к компании является сотрудником (employee), а компания по отношению к человеку – работодателем (employer). Использование имен полюсов ассоциаций не является обязательным, но чаще всего оказывается проще указать имена полюсов вместо имен ассоциаций или, по крайней мере, вместе с ними.

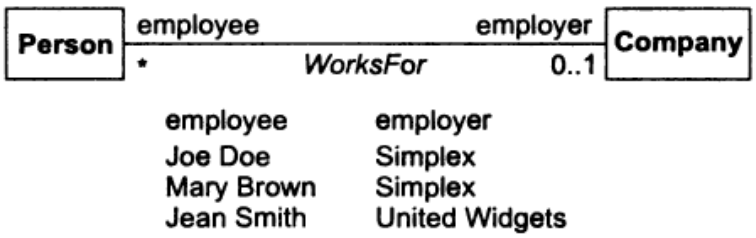

Рис. 4.7 – Отображение имен полюсов ассоциации в нотации UML

Имена полюсов ассоциаций особенно удобны для прослеживания ассоциаций, потому что каждый из них может рассматриваться как псевдоатрибут. Каждый полюс бинарной ассоциации ссылается на объект или множество объектов, связанных с исходным объектом. С точки зрения исходного объекта, прослеживание ассоциации – это операция, которая возвращает связанные с ним объекты. Имя полюса ассоциации – это средство для прослеживания ассоциации без явного ее указания.

Имена полюсов позволяют унифицировать несколько ссылок на один и тот же класс. При построении диаграмм классов следует корректно использовать имена полюсов ассоциаций и не вводить отдельный класс для каждой ссылки. На рис. 4.8 *Человек* с *Ребенком* может быть представлен двумя экземплярами: один для родителя и один для ребенка. В корректной модели экземпляр *Челове-*

ка принимает участие в двух и более связях: дважды в качестве родителя и произвольное количество раз в качестве ребенка. В корректной модели нужно показать, что ребенок не обязательно должен иметь родителя, чтобы рекурсия могла быть прервана.

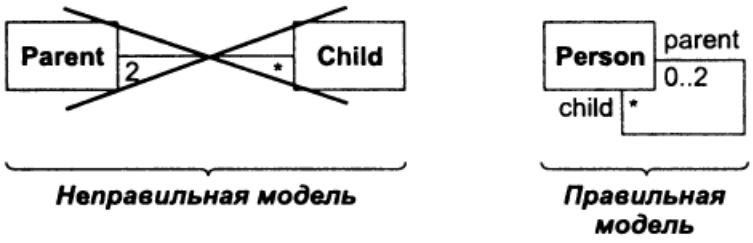

Рис. 4.8 – Моделирование ссылок на один и тот же класс

Поскольку имена полюсов ассоциации позволяют отличать объекты друг от друга, все имена на дальнем полюсе ассоциации, прикрепленной к некоторому классу, должны быть уникальными. Хотя имя ставится около целевого объекта ассоциации, фактически оно является псевдоатрибутом исходного класса, а потому должно быть уникально внутри него. По той же причине имя полюса ассоциации не должно совпадать с именем какого-либо атрибута исходного класса.

Подобно тому, как объекты класса могут быть описаны при помощи атрибутов, связи ассоциации также могут быть описаны атрибутами. UML позволяет представлять информацию такого характера при помощи классов ассоциаций. Класс ассоциаций – это ассоциация, которая одновременно является классом. Подобно связям ассоциаций, экземпляры класса ассоциаций обладают индивидуальностью, связанной с теми объектами, между которыми они проводятся. Подобно обычным классам, классы ассоциаций могут иметь атрибуты и операции и участвовать в ассоциациях. Классы ассоциаций присутствуют в формулировке задачи в виде наречий или получаются абстрагированием известных значений.

На рис. 4.9 *accessPermission* (*разрешениеДоступа*) является атрибутом *AccessibleBy* (*Доступно*). Данные в нижней части рисунка показывают возможные

значения каждой связи. В UML класс ассоциации обозначается прямоугольником, прикрепленным к ассоциации пунктирной линией.

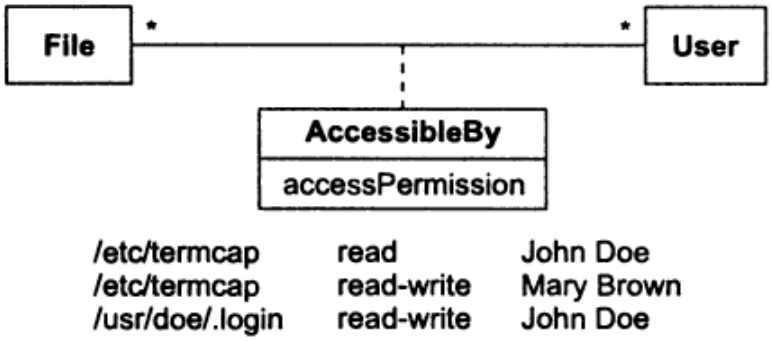

Рис. 4.9 – Атрибуты связи ассоциаций

Основанием для введения классов ассоциаций послужила возможность создания ассоциаций многие-ко-многим. Атрибуты таких ассоциаций без всяких сомнений являются принадлежностью связей и не могут быть приписаны ни к одному из объектов-участников. На рис. 4.9 атрибут *accessPermission* относится к файлу и пользователю одновременно и не может быть прикреплен только к одному из них без потери информации.

Квалифицированной называется ассоциация, у которой имеется специальный атрибут (квалификатор), используемый для того, чтобы отличать друг от друга объекты, находящиеся на полюсе ассоциации с кратностью «много». Квалификаторы могут быть определены для ассоциаций типа один-ко-многим и многие-ко-многим. Квалификатор позволяет выбрать отдельный объект из множества целевых объектов, уменьшая таким образом эффективную кратность до значения «один». Квалифицированная ассоциация с целевой кратностью «один» или «не более одного» образует четкий маршрут для поиска целевого объекта по исходному.

На рис. 4.10 демонстрируется наиболее типичный пример использования квалификатора для ассоциаций с кратностью один-ко-многим. *Банк обслуживает* множество *счетов*. *Счет принадлежит* одному единственному *банку*. В контексте банка уникальный счет определяется своим номером. *Банк* и *Счет* –

это классы, а НомерСчета – квалификатор. Квалификация уменьшает эффективную кратность ассоциации до единицы.

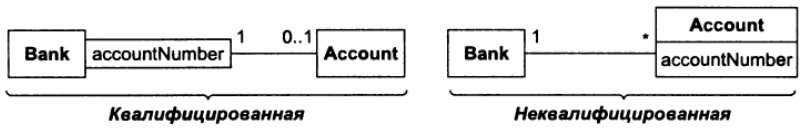

Рис. 4.10 – Квалифицированная ассоциация

Обе модели (квалифицированная и неквалифицированная) вполне корректны, но модель с квалифицированной ассоциацией сообщает дополнительную информацию. В квалифицированной модели добавляется ограничение на кратность: сочетание банка и номера счета дает не более одного счета. Квалифицированная модель также передает значение номера счета для прослеживания модели, что отражается в методах. Сначала следует найти банк, а затем указать номер счета, и в результате вы получите нужный счет.

Для обозначения квалификатора используется небольшой прямоугольник, который пристыковывается к исходному классу около конца линии, обозначающей ассоциацию. Квалификатор может быть пристыкован к любой стороне прямоугольника исходного класса. Исходный класс вместе с квалификатором определяют целевой класс.

4.3.3 Обобщение и наследование

Обобщение — это отношение между классом (суперклассом) и одной или несколькими его вариациями (подклассами). Обобщение объединяет классы по их общим свойствам, благодаря чему обеспечивается структурирование описания объектов. Суперкласс характеризуется общими атрибутами, операциями и ассоциациями. Подклассы добавляют к ним свои собственные атрибуты, операции и ассоциации. Говорят, что подкласс наследует составляющие суперкласса. Обобщение иногда называется отношением типа «является», поскольку каждый экземпляр подкласса одновременно *является* экземпляром суперкласса.

Простое обобщение упорядочивает классы в рамках некоторой иерархии. В этом случае каждый подкласс имеет одного непосредственного предка (его суперкласс). Уровней обобщения может быть много.

Примеры обобщений приведены на рис. 4.11. Оборудование может быть насосом, теплообменником или резервуаром. Насосы бывают нескольких типов: центробежные, мембранные и плунжерные. Резервуары бывают сферические, с наддувом и с плавающей крышей. То, что символ обобщения резервуара изображен ниже символа обобщения насоса, никакого особого значения не имеет. В нижней части рисунка изображены несколько объектов. Каждый объект наследует составляющие от одного класса с каждого уровня иерархии обобщений. Поэтому объект P101 обладает составляющими оборудования, насоса и мембранного насоса. Объект E302 обладает составляющими оборудования и теплообменника.

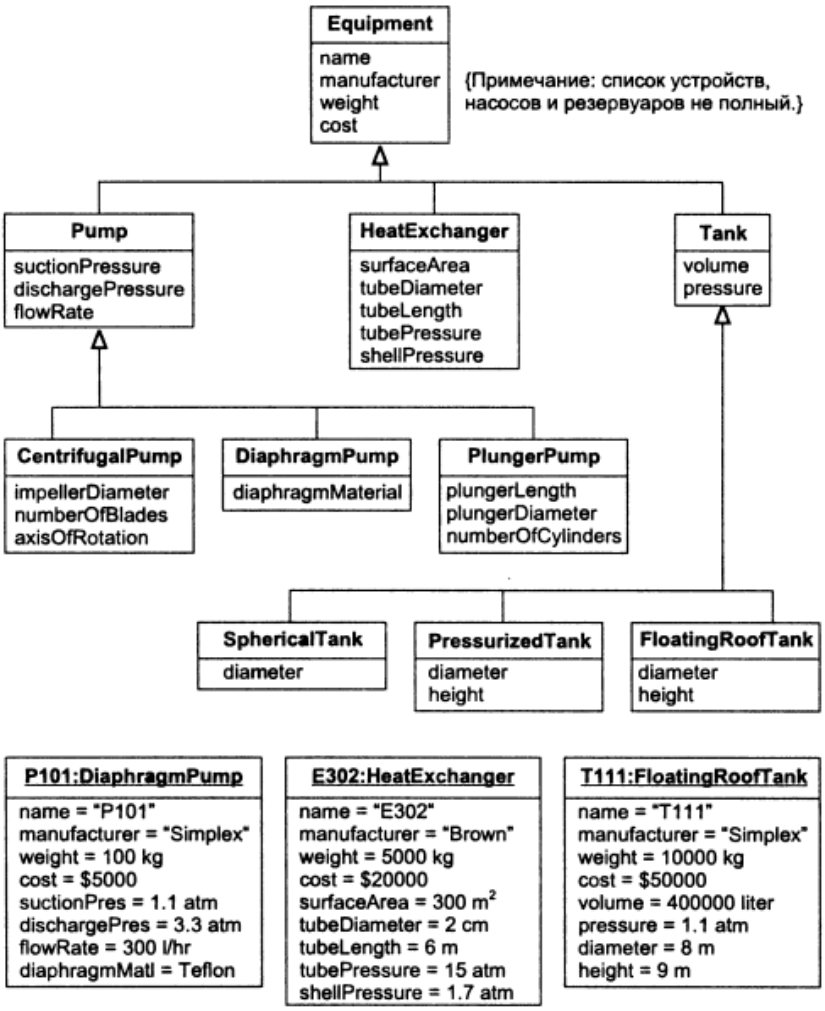

Рис. 4.11 – Многоуровневая иерархия наследования с экземплярами

Обобщение обозначается большой незакрашенной стрелкой. Стрелка указывает на суперкласс. Суперкласс можно соединять с каждым из его подклассов непосредственно, но предпочтительнее будет группировать обобщения в дерево. Треугольник можно повернуть и расположить его с любой стороны от суперкласса, но, по возможности, следует изображать суперкласс сверху, а его подклассы — снизу. Комментарии UML, указывающие на наличие дополнительных подклассов, не приведенных на диаграмме, ставятся в фигурных скобках.

Обобщение транзитивно и действует через произвольное количество уровней иерархии. Термины *предок* и *потомок* используются для описания классов, находящихся далеко друг от друга по уровням, но связанных отношением обобщения. Экземпляр подкласса одновременно является экземпляром всех его

предков. Экземпляр обладает значениями всех атрибутов всех классов-предков. Экземпляр может вызывать любую операцию, указанную у любого из его предков. Подклассы не только наследуют все составляющие своих предков, но и добавляют к ним свои собственные составляющие.

Слово, написанное на диаграмме рядом с линией, обозначающей обобщение, – это имя набора обобщений. Имя набора обобщений — это перечислимый атрибут, показывающий, какой аспект объекта абстрагируется конкретным обобщением. Каждый набор должен абстрагировать только один аспект. Значения наборов обобщений находятся во взаимно-однозначном соответствии с подклассами обобщений. Название набора обобщений указывать не обязательно.

Обобщение служит трем основным целям. Первая цель — обеспечение поддержи полиморфизма. Операция может быть вызвана на уровне суперкласса, а компилятор объектно-ориентированного языка автоматически разрешит вызов в метод, соответствующий классу вызывающего объекта. Полиморфизм увеличивает гибкость программного обеспечения: вы добавляете новый подкласс и автоматически наследуете поведение суперкласса. Более того, новый подкласс не нарушает работу существующего кода.

Вторая цель обобщения состоит в структурировании описаний объектов. Используя обобщение, вы делаете концептуальное утверждение, образуя таксономию и упорядочивая объекты на основании их сходств и различий. Такой подход гораздо более основателен, нежели моделирование каждого класса в изоляции от всех остальных.

Третья цель состоит в обеспечении повторного использования кода: вы можете наследовать код в рамках одного приложения, а также из существующих приложений (библиотек классов). Повторное использование повышает производительность по сравнению с многократным переписыванием кода «с нуля». Обобщение позволяет корректировать код для получения именно такого поведения, которое требуется в данном приложении.

Подкласс может подменять или перекрывать составляющую суперкласса, определяя составляющую с тем же именем внутри себя. Подмена составляющих может потребоваться по нескольким причинам: для спецификации поведения, зависящего от подкласса; для уточнения спецификации составляющей; для повышения производительности.

Подменять можно методы и значения по умолчанию для атрибутов. Никогда не следует подменять сигнатуру составляющей. Подмена должна сохранять тип атрибута, количество и тип аргументов операции, а также тип возвращаемого операцией значения. Уточнение типа атрибута или аргумента операции (то есть указание в качестве типа подтипа исходного типа) является формой ограничения и должно использоваться с осторожностью.

Вопросы для самоконтроля:

1. Что такое объект и класс в объектно-ориентированном проектировании?
2. Как отображаются атрибуты и операции класса в нотации UML?
3. Что такое связь и ассоциация в объектно-ориентированном проектировании?
4. Что такое кратность связи?
5. Могут ли связи ассоциаций быть описаны атрибутами?
6. Какая ассоциация называется квалифицированной?
7. Что такое обобщение?
8. Какие цели преследует обобщение?

4.4 Моделирование состояний

4.4.1 События

Событие — это происшествие, случившееся в определенный момент времени, например, *нажатие левой кнопки мыши*. Часто события соответствуют глаголам в прошедшем времени (питание было включено, будильник был уста-

новлен) или выполнению некоторого условия (опустошился лоток для бумаги) в описании задачи. По определению, событие происходит мгновенно, по крайней мере, во временном масштабе приложения. Событие рассматривается как атомарное и скоротечное происшествие. Неявным атрибутом события является момент его осуществления. Продолжительные изменения, осуществляющиеся в течение некоторого промежутка времени, хорошо описываются с помощью концепции состояния.

Одно событие может логически предшествовать другому или следовать за ним. События могут быть и несвязанными друг с другом. Несвязанные события называются параллельными. Они никак не влияют друг на друга. Если временная задержка при передаче информации между двумя точками превышает временной интервал между событиями, эти события обязаны быть параллельными, поскольку они никак не могут повлиять друг на друга. При моделировании системы не следует задавать относительный порядок параллельных событий, потому что на практике они могут происходить в любом порядке.

К событиям относятся не только нормальные происшествия, но и ошибочные ситуации. Ошибочная ситуация ничем не отличается от любого другого события. Только в нашей интерпретации она становится «ошибкой».

Термин «событие» часто используется в нескольких смыслах. В некоторых случаях он обозначает экземпляр, а в других — класс. На практике эта двусмысленность обычно не создает проблем, так как точный смысл слова следует из контекста. При необходимости можно употреблять точные термины: «осуществление события» и «происшествие» (экземпляр события) и «тип события» (класс). События бывают разных видов. Чаще всего встречаются события сигналов, события изменения и события времени.

Сигнал — это явная односторонняя передача информации от одного объекта к другому. Объект, передающий сигнал другому объекту, может рассчитывать на получение ответа, но этот ответ будет отдельным сигналом, и его отправка (или задержка) будет целиком зависеть от второго объекта.

Событие сигнала — это событие получения или отправки сигнала. Обычно более важным считается получение сигнала, потому что оно влияет на объект получатель. Обратите внимание на разницу между сигналом и событием сигнала: сигнал — это сообщение между объектами, а событие сигнала — это происшествие.

Каждая передача сигнала является уникальным происшествием, но мы группируем их в классы сигналов и даем каждому классу имя, подчеркивая общую структуру и поведение. Например, *вылет рейса самолета № 123 из аэропорта 10 октября* — это экземпляр класса сигналов *ВылетРейса.* Некоторые сигналы являются обычными происшествиями, но большинство из них характеризуются атрибутами, в которых хранятся передаваемые этими сигналами значения.

Например, на рис. 4.12 класс *FlightDeparture (ВылетРейса)* обладает атрибутами *airline (авиакомпания), flightNumber (номерРейса), city (город), date (дата)*. В UML сигнал обозначается ключевым словом «signal» в угловых кавычках («»), которое ставится над именем класса сигнала в верхнем разделе прямоугольника. Во втором разделе указываются атрибуты сигнала.

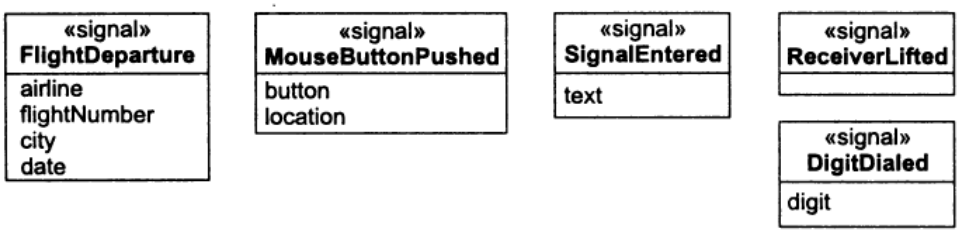

Рис. 4.12 – Классы сигналов и их атрибуты

Событие изменения — это событие, вызванное выполнением логического выражения. Суть события состоит в том, что некоторое выражение постоянно проверяется, и как только его значение изменяется с «ложно» на «истинно», осуществляется событие изменения.

В UML событие изменения обозначается ключевым словом «when», за которым следует логическое выражение в круглых скобках. На рис. 4.13 приведено несколько примеров событий изменения.

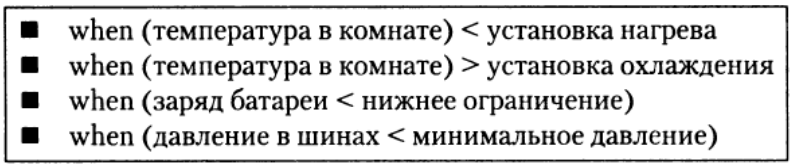

Рис. 4.13 – События изменения

Событие времени — это событие, вызванное достижением момента абсолютного времени или истечением временного интервала. В UML момент абсолютного времени обозначается ключевым словом «when», за которым следует временное выражение в круглых скобках. Временной интервал обозначается ключевым словом «after», за которым следует выражение, результатом вычисления которого является временной интервал (см. рис. 4.14).

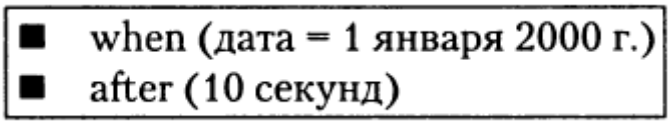

Рис. 4.14 – События времени

4.4.2 Состояния

Состояние — это абстракция значений и связей объекта. Множества значений и связей группируются в состояние в соответствии с массовым поведением объектов. Например, состояние банка может быть «платежеспособный» или «банкрот», в зависимости от того, что больше: активы или обязательства. Состояния часто соответствуют отглагольным формам или деепричастиям (*Ожидает, Дозванивается*) или выполнению некоторого условия (*Включен, НижеТочкиЗамерзания*).

На рис. 4.15 показана система обозначений состояния в UML: прямоугольник со скругленными углами, в котором ставится необязательное название состояния. Мы выделяем название состояния полужирным шрифтом, центрируем его и пишем с заглавной буквы.

Рис. 4.15 – Состояния

Определяя состояния, мы не учитываем атрибуты, не оказывающие влияния на поведение объекта, и объединяем вместе в одно состояние все комбинации значений и связей, характеризующиеся одинаковыми откликами на события. Разумеется, каждый атрибут должен влиять на поведение, иначе он не будет иметь никакого значения, однако, достаточно часто некоторые атрибуты не влияют на последовательность управления. Их можно рассматривать просто как значения параметров состояния.

Объекты класса обладают конечным числом возможных состояний. В конкретный момент времени каждый объект может находиться ровно в одном состоянии. Объекты могут проходить через одно или несколько состояний в течение времени своего существования. В конкретный момент времени разные объекты класса могут охватывать широкий спектр состояний.

Состояние описывает отклик объекта на получаемые события. В конкретном состоянии игнорируются любые события, за исключением тех, поведение при получении которых описано явным образом. Отклик на событие может быть вызовом поведения или изменением состояния. Например, если в состоянии *Гудок* нажать кнопку с цифрой на телефоне, зуммер сбрасывается, а телефонная линия переходит в состояние *НаборНомера*. Если же в состоянии *Гудок* повесить трубку, линия отключается и переходит в состояние *Свободно*.

Между событиями и состояниями существует некоторая симметрия. События — это точки на линии времени, а состояния — интервалы. Состояние соот-

ветствует интервалу между двумя точками, обозначающими два полученных объектом события. Например, между снятием трубки и нажатием первой цифры, телефонная линия находится в состоянии *Зуммер*. Состояние объекта зависит от предыдущих событий, которые в большинстве случаем перекрываются последующими событиями. Например, события, произошедшие до того, как трубка была повешена, не влияют на будущее поведение. Состояние *Свободно* «забывает» о событиях, полученных до сигнала *повесить трубку*.

События и состояния зависят от уровня абстрагирования. Например, коммивояжер, планирующий свой маршрут, будет рассматривать каждый его сегмент как отдельное событие. Информатор в аэропорту будет объявлять о вылетах и прибытиях. Авиадиспетчерская служба разделить каждый перелет на множество географических отрезков.

Состояние можно характеризовать несколькими способами. Рисунок 4.16 демонстрирует это для состояния *Звонок будильника* для часов. Состояние обладает именем и описанием на естественном языке. Последовательность событий, которая приводит к этому состоянию, состоит из установки будильника, произвольных действий, не приводящих к его сбрасыванию, и наступления заданного момента времени. Условие состояния выражается в терминах параметров, таких как текущее и целевое время. Звонок прекращается после 20 секунд. Таблица событий и откликов показывает реакцию на события *текущее время* и *нажатие кнопки*. В этой таблице указывается не только действие, но и следующее состояние. Разные описания состояния могут перекрываться.

Состояние: *AlarmRinging*
Описание: alarm on watch is ringing to indicate target time
Событие, приводящее к данному состоянию:
setAlarm (*targetTime*)
any sequence not including *clearAlarm*
when (*currentTime* = *targetTime*)
Условие, характеризующее данное состояние:
alarm = on, alarm set to *targetTime*, *targetTime* J *currentTime* J
targetTime + 20 seconds, and no button has been pushed since *targetTime*
События, возможные в данном состоянии:

Событие	Отклик	Следующее состояние
when (*currentTime* = *targetTime* + 20)	*resetAlarm*	*normal*
buttonPushed (any button)	*resetAlarm*	*normal*

Рис. 4.16 – Разные способы описать состояние

4.4.3 Переходы и условия

Переход — это мгновенная смена одного состояния другим. Например, когда вы отвечаете на входящий звонок, телефонная линия переходит из состояния *Звонок* в состояние *Разговор*. Говорят, что переход запускается при смене исходного состояния целевым. Исходное и целевое состояния обычно отличаются друг от друга, но могут и совпадать. Переход запускается, когда происходит связанное с ним событие (если только необязательное сторожевое условие не приводит к игнорированию события). Выбор целевого состояния зависит как от исходного состояния, так и от полученного события. Событие может вызвать переходы во множестве объектов. С концептуальной точки зрения эти переходы происходят одновременно.

Сторожевое условие — это логическое выражение, которое должно быть истинным, чтобы переход мог запуститься. Например, сигнал светофора на перекрестке может переключиться только в том случае, если на дороге имеются ожидающие этого машины. Переход со сторожевым условием запускается в тот момент, когда осуществляется соответствующее событие, но только если в этот же момент выполнено его сторожевое условие. Например, «когда будешь вы-

ходить из дома (событие), если на улице будет ниже нуля (условие) — надень перчатки (целевое состояние)». Сторожевое условие проверяется только один раз, в тот момент, когда осуществляется событие, и если условие выполняется — происходит переход. Обратите внимание, что сторожевое условие концептуально отличается от события: условие проверяется только один раз, тогда как наличие события, по сути, проверяется непрерывно.

На рис. 4.17 показаны переходы со сторожевыми условиями для светофоров на перекрестке. Одна пара фотоэлементов контролирует полосы в направлении север-юг из которых возможен поворот налево. Другая пара контролирует полосы в направлении запад-восток из которых тоже возможен поворот налево. Если на одной из пар полос отсутствуют машины, управляющая логика светофора пропускает часть цикла, разрешающую левый поворот.

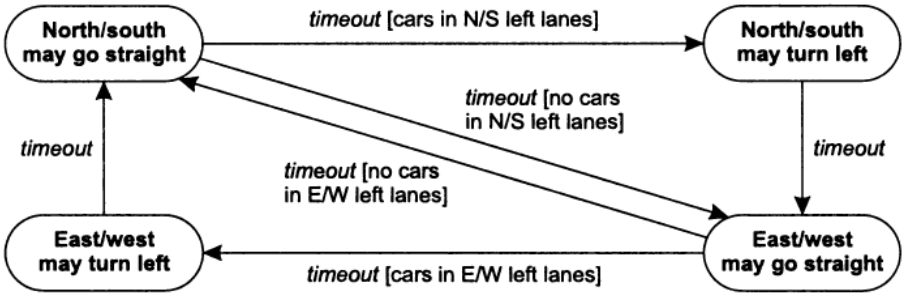

Рис. 4.17 – Переходы со сторожевыми условиями

В UML для обозначения перехода используется линия, соединяющая исходное состояние с целевым. на одном из концов линии, указывающее на целевое состояние, ставится стрелка. Линия может состоять из нескольких сегментов. Событие может быть указано в качестве метки перехода. После события в квадратных скобках можно указать необязательное сторожевое условие. Название события выделяется курсивом, а сторожевое условие записывается в квадратных скобках.

4.4.4 Диаграммы состояний

Диаграмма состояний — это граф, узлами которого являются состояния, а направленными дугами — переходы между состояниями. Диаграмма состояний описывает последовательности состояний, вызываемые последовательностями событий. Названия состояний должны быть уникальными в рамках диаграммы. Все объекты класса следует диаграмме состояний, описывающей общее для них поведение. Диаграмма состояний может быть реализована непосредственной интерпретацией или преобразованием семантики в эквивалентный программный код.

Модель состояний состоит из множества диаграмм состояний, по одной на каждый класс, поведение которого с течением времени важно для приложения. Диаграммы состояний должны быть согласованы по интерфейсам (событиям и сторожевым условиям). Отдельные диаграммы взаимодействуют друг с другом посредством передачи событий, а также косвенно, через сторожевые условия. Некоторые события и сторожевые условия присутствуют только на одной диаграмме, тогда как другие — на нескольких.

Класс, имеющий несколько состояний, характеризуется важным поведением во времени. Если же класс обладает одним состоянием, его поведение во времени можно игнорировать. Диаграммы состояний с одним состоянием можно описать в простой форме без всякой графики, а именно в виде таблицы воздействий и откликов, в которой будут приводиться события и сторожевые условия, а также вызываемое ими поведение.

На рис. 4.18 показана диаграмма состояний для телефонной линии. Данная диаграмма относится именно к телефонной линии, а не звонящему или вызываемому абоненту. На диаграмме приведены последовательности, описывающие нормальные звонки, а также некоторые ненормальные последовательности, например, тайм-аут при наборе номера или перегрузка линий. Для обозначения диаграммы состояний в UML используется прямоугольник. Название диаграм-

мы указывается в пятиугольном теге в левом верхнем углу. Внутри прямоугольника изображаются состояния и переходы, образующие диаграмму.

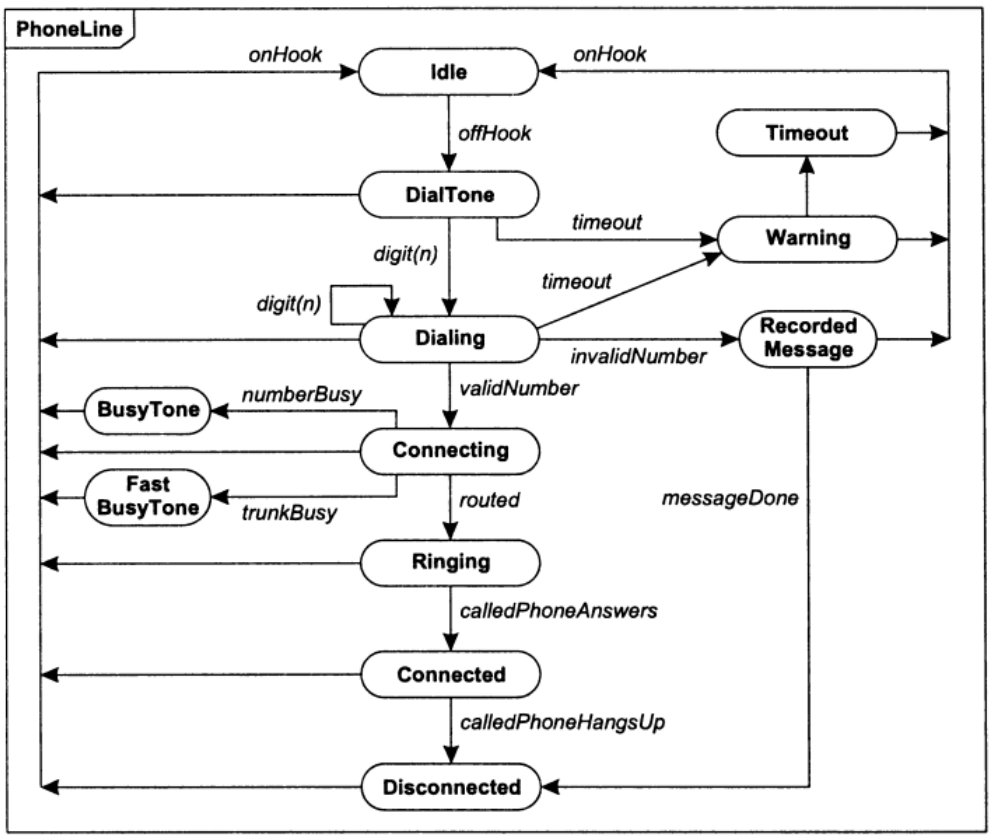

Рис. 4.18 – Диаграмма состояний телефонной линии

Перед началом вызова телефонная линия находится в отключенном состоянии (состоянии ожидания). Когда трубка снимается с рычага, она издает сигнал ответа станции (гудок). После этого можно набирать цифры. При вводе корректного номера система пытается выполнить соединение и направить его к нужному адресату. Соединение может оказаться невозможным, если абонент или его узел заняты. если же соединение оказывается успешным, телефон вызываемого абонента начинает звонить. Если абонент отвечает на звонок, оба абонента могут осуществить разговор. Когда один из абонентов вешает трубку, линия разъединяется и снова возвращается в состояние ожидания.

Состояния не определяют все значения объекта полностью. Например, состояние *Dialing (НаборНомера)* включает все последовательности неполных телефонных номеров. Не обязательно рассматривать отдельные номера как раз-

ные состояния, потому что они характеризуются одинаковым поведением. Реальный набранный номер следует хранить как атрибут.

Если из состояния выходит несколько переходов, первое осуществившееся событие запускает соответствующий ему переход. Если происходит событие, которому не сопоставлен ни один переход, оно просто игнорируется. Если событию соответствует несколько переходов, выбран будет только один из них, и притом случайным образом.

Диаграммы состояний могут описывать непрерывные циклы или одноразовые жизненные циклы. Диаграмма состояний телефонной линии является непрерывным циклом. Описывая обычные режимы использования телефона, мы не интересуемся тем, каким образом цикл был запущен.

Одноразовые диаграммы состояний описывают объекты с конечным сроком существования. Такие диаграммы имеют начальное и конечное состояния. Сразу после создания объект оказывается в начальном состоянии. Вход в конечное состояние означает уничтожение объекта. На рис. 4.19 показан упрощенный жизненный цикл игры в шахматы с начальным состоянием по умолчанию (сплошной кружок) и конечным состоянием по умолчанию («бычий глаз»).

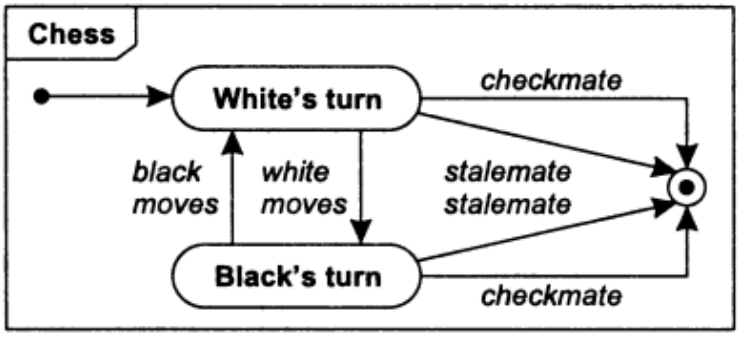

Рис. 4.19 – Диаграмма состояний игры в шахматы

Начальное и конечное состояния можно обозначать точками входа и выхода. На рис. 4.20 точка входа *Start* соединена с первым ходом белых, и игра заканчивается одним из трех возможных состояний. Точки входа (пустые круж-

ки) и выхода (кружки с символом X) ставятся на периметре диаграммы состояний и могут иметь имена.

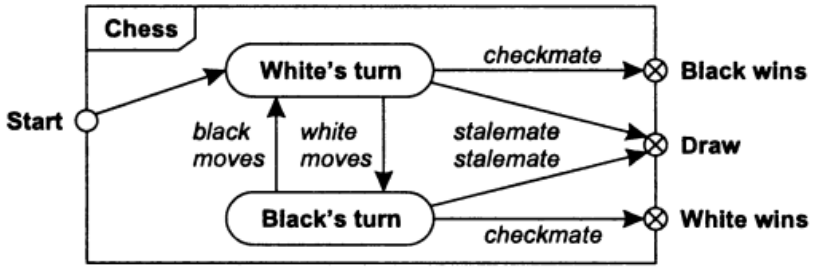

Рис. 4.20 – Диаграмма состояний игры в шахматы с точками входа и выхода

4.4.5 Поведение на диаграммах состояний

Диаграммы состояний были бы не слишком полезны, если бы они описывали только события. Полное описание объекта должно указывать, что именно делает объект в ответ на события.

Действие — это ссылка на поведение, выполняемое в ответ на произошедшее событие. Деятельность — это фактическое поведение, которое может вызываться любым количеством действий. Например, деятельность *разъединить-Линию* может выполняться при переходе, при входе в состояние или при выходе из него, а также при наступлении какого-либо иного события в состоянии.

Деятельность может описывать внутренние управляющие операции, например, установку атрибутов или порождение других событий. Эта деятельность не имеет аналогов в реальном мире и предназначена для структурирования управления при реализации. Например, программа может увеличивать внутренний счетчик на единицу каждый раз при осуществлении какого-либо события.

Деятельность обозначается косой чертой (/), после которой ставится название или описание деятельности. Деятельность указывается после вызывающего ее события. Ключевое слово «do» используется для обозначений текущей деятельности и не может использоваться в качестве имени события. На рис. 4.21

показана диаграмма состояний для всплывающего меню рабочей станции. При нажатии правой кнопки мыши меню отображается на экране. Когда пользователь отпускает эту кнопку, меню исчезает. Пока меню отображается на экране, в нем подсвечивается один элемент, над которым в данным момент находится указатель мыши.

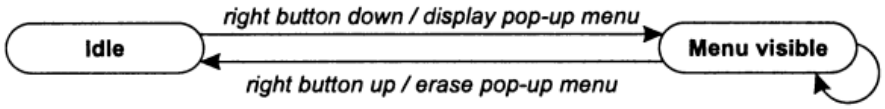

Рис. 4.21 – Деятельность для всплывающего меню

Текущей называется деятельность, занимающая некоторый промежуток времени. По определению, такая деятельность может выполняться только в некотором состоянии и не может прикрепляться к переходу. Например, индикатор аварии у ксерокса может мигать в состоянии *Затор бумаги* (см. рис. 4.22). Текущая деятельность включает непрерывные операции, такие как отображение картинки на телеэкране, а также последовательные операции, завершающиеся по прошествии некоторого промежутка времени (например, закрытие клапана).

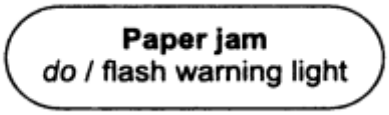

Рис. 4.22 – Текущая деятельность для ксерокса

Для обозначения текущей деятельности используется ключевое слово «do» и символ косой черты. Текущая деятельность может быть прервана событием, полученным в процессе выполнения этой деятельности. Это событие может вызвать переход из состояния, в котором осуществляется текущая деятельность, но может и не вызвать такого перехода. Например, робот, перемещающий деталь, может столкнуться с сопротивлением, что приведет к остановке его движения.

Деятельность может быть прикреплена не только к переходу, но и ко входу в состояние или к выходу из него. Никаких отличий в возможностях двух систем обозначений нет, однако, часто при всех переходах в одно и то же состояние выполняется одинаковая деятельность, которую, в таком случае, удобнее привязать к самому событию.

Часто единственным назначением состояния является последовательное выполнение некоторой деятельности. Как только деятельность завершается, запускается переход в следующее состояние. Стрелка без названия события обозначает автоматический переход, который запускается, как только завершается деятельность, связанная с исходным состоянием. Такой переход называется переходом по завершении, потому что он переключается завершением деятельности в исходном состоянии.

Сторожевое условие проверяется только один раз, в тот момент, когда осуществляется событие. Если переход имеет переход по завершении, а его сторожевое условие не выполнено, состояние остается активным и может привести к сбою управления: событие завершения деятельности не может произойти второй раз. Если состояние имеет переходы по завершении, сторожевые условия должны охватывать весь спектр возможных событий. Специальное условие «else» позволяет задать ситуацию, когда все остальные условия оказываются ложными. Не используйте сторожевые условия на переходы по завершении для ожидания изменения значения. Лучше ожидать события изменения.

Вопросы для самоконтроля:

1. Что такое событие?
2. Что такое событие сигнала?
3. Что такое событие изменения?
4. Что такое событие времени?
5. Как обозначается состояние в стандарте UML?
6. Что такое переход?
7. Какие условия называются сторожевыми?

8. Что изображается на диаграмме состояний?

9. Какая деятельность называется текущей?

4.5 Моделирование взаимодействий

4.5.1 Модели вариантов использования

Действующее лицо — это непосредственный внешний пользователь системы. Это объект или множество объектов, непосредственно взаимодействующих с системой, но не являющихся ее частью. Каждое действующее лицо является обобщением группы объектов, ведущих себя определенным образом по отношению к системе. Например, клиент и ремонтник являются разными действующими лицами по отношению к торговому автомату. Действующими лицами по отношению к системе бюро путешествий могут быть путешественники, агенты и авиакомпании. По отношению к компьютерной базе данных ими могут быть пользователи и администраторы. Действующими лицами могут быть люди, устройства и другие системы — все, что взаимодействует с интересующей нас системой непосредственно.

Объект может быть связан с несколькими действующими лицами, если его поведение обладает разными гранями. Например, объекты Мэри, Фрэнк и Пол могут быть клиентами торгового автомата. Пол может одновременно являться и ремонтником, обслуживающим этот автомат.

Действующее лицо должно иметь одну четко определенную цель. Объекты и классы обычно сочетают в себе несколько назначений (целей). Действующее же лицо представляет лишь одну грань объекта в его взаимодействии с системой. Одно и то же лицо может представлять объекты разных классов, ведущие себя одинаково по отношению к системе. Например, люди, так или иначе взаимодействующие с торговым автоматом, могут быть разделены на две основные категории: клиенты и ремонтники. Каждое действующее лицо обладает согласованным набором возможностей тех объектов, которые оно представляет.

Моделирование действующих лиц помогает определить границы системы, то есть идентифицировать объекты, находящиеся внутри системы, и объекты, лежащие на ее границе. Действующее лицо непосредственно взаимодействует с системой. Объекты, участвующие только в косвенных взаимодействиях с системой, не являются действующими лицами и потому не должны включаться в модель системы. Любое косвенное взаимодействие должно осуществляться посредством действующих лиц. Например, диспетчер мастеров по ремонту торговых автоматов не является действующим лицом по отношению к этим автоматам. Непосредственно с автоматами работают только мастера. Если нужно построить модель косвенного взаимодействия внешних объектов, нужно строить модель среды как большой системы, включающей в себя исходную систему. Например, можно построить модель ремонтной службы, включающей в себя (в качестве действующих лиц) диспетчеров, мастеров и торговые автоматы, но эта модель уже не будет моделью торговых автоматов.

Различные взаимодействия действующих лиц с системой группируются в варианты использования. Вариант использования — это связный элемент функциональности, представляемый системой при взаимодействии с действующими лицами. Например, лицо *Клиент* может *купить напиток* в торговом автомате. Клиент кидает монеты в автомат, выбирает продукт и забирает свой напиток. *Ремонтник* может *провести техническое обслуживание* автомата. На рис. 4.23 приведены несколько вариантов использования для торгового автомата.

- **Купить напиток.** Торговый автомат выдает напиток после того, как клиент выбирает нужный вариант и платит за него.
- **Провести плановый ремонт.** Ремонтник выполняет плановое техобслуживание автомата, необходимое для обеспечения его безотказной работы.
- **Провести техническое обслуживание.** Ремонтник выполняет незапланированное обслуживание автомата при выходе его из строя.
- **Загрузить продукты.** Обслуживающий персонал загружает продукты в торговый автомат для пополнения запасов продаваемых напитков.

Рис. 4.23 – Варианты использования торгового автомата

В каждом варианте использования участвуют одно или несколько действующих лиц и система. В варианте использования *купить напиток* участвует лицо *Клиент,* а в варианте использования *провести плановый ремонт* участвует лицо *Работник*. Вариант использования подразумевает обмен последовательностью сообщений между системой и действующими лицами. Например, в варианте использования *купить напиток Клиент* сначала вставляет монету, а торговый автомат отображает текущий аванс. Эта процедура может быть повторена несколько раз. Затем клиент нажимает кнопку выбора товара, а автомат выдает заказанный товар и сдачу, если таковая имеется.

Некоторые варианты использования характеризуются фиксированной последовательностью сообщений. Однако чаще последовательность сообщений может варьироваться в определенных пределах. Например, клиент может кинуть в автомат некоторое количество монет (вариант использования *купить напиток)*. В зависимости от количества и номинала монет, а также от выбранного товара автомат может либо вернуть сдачу, либо нет. Вариации последовательностей можно показать, продемонстрировав несколько примеров различающихся между собой последовательностей поведения. Обычно сначала определяется основная последовательность, а затем — необязательные последовательности, повторения и другие вариации.

Частью варианта использования являются и сбойные ситуации. Например, если клиент выбирает напиток, который уже кончился, торговый автомат отображает соответствующее сообщение. Кроме того, клиент может отменить транзакцию покупки. Например, он может нажать кнопку возврата монет в любой до того, как его выбор будет принят. В этом случае автомат возвращает монеты клиенты, и последовательность поведения для варианта использования завершается. С точки зрения пользователя некоторые виды поведения могут считаться ошибками. Однако проектировщик должен учитывать все возможные последовательности. С точки зрения системы ошибки пользователя или ресурсов являются разновидностями поведения, которые устойчивая система должна быть способна обработать.

Вариант использования объединяет все поведение, имеющее отношение к элементу функциональности системы: нормальное поведение, вариации нормального поведения, исключительные ситуации, сбойные ситуации и отмены запросов. На рис. 4.24 представлено подробное описание варианта использования покупка напитка. Объединение нормального и аномального поведения в одном варианте использования помогает убедится, что все последовательности взаимодействия рассматриваются вместе. В полной модели варианты использования образуют разбиение функциональности системы. Все варианты должны находиться на сравнимом уровне абстракции.

Вариант использования: Покупка напитка.
Краткое описание: Торговый автомат выдает напиток после того, как клиент выбирает нужный вариант и платит за него.
Действующие лица: Клиент.
Исходные условия: Автомат ожидает опускания монеты.
Описание: Автомат изначально находится в состоянии ожидания и выводит на дисплей сообщение «Опустите монеты». Клиент опускает монеты в щель автомата. Автомат выводит на дисплей принятую от клинта сумму и включает подсветку кнопок с названиями напитка. Автомат выдает соответствующий напиток и выдает сдачу, если напиток стоит меньше, чем заплатил клиент.
Исключения:
Отмена: Если клиент нажмет кнопку отмены до того, как произведет выбор напитка, автомат вернет клиенту деньги и перейдет в состояние ожидания.
Напиток отсутствует: Если клиент выбирает напиток, который в данный момент отсутствует в автомате, выводится сообщение: «Напиток отсутствует». Автомат готов к приему монет и выбору напитка клиентом.
Недостаточно денег: Если клиент выбирает напиток, который стоит больше, чем он заплатил, выводится сообщение: «Необходимо доплатить *пп.пп* руб. для покупки этого напитка», где *пп.пп* — недостающая сумма. Автомат продолжает принимать монеты и готов к выбору напитка клиентом.
Нет сдачи: Если клиент вставил достаточное количество монет для покупки напитка, но в автомате нет денег, чтобы корректно выдать сдачу, выводит сообщение «Невозможно выдать сдачу». Автомат продолжает принимать монеты и готов к выбору напитка клиентом.
Постусловия: Автомат готов к приему монет.

Рис. 4.24 – Описание варианта использования

Любая система обладает своим множеством вариантов использования и множеством действующих лиц. Каждый вариант использования описывает элемент предоставляемой системой функциональности. Множество вариантов использования описывает всю функциональность системы на некотором уровне

абстракции. Каждое действующее лицо представляет собой один вид объектов, для которых система может выполнять некоторое поведение. Множество действующих лиц описывает полное множество объектов, которые могут быть обслужены системой. Объекты аккумулируют поведение всех систем, с которыми они взаимодействуют в качестве действующих лиц.

Язык UML предусматривает систему графических обозначений для вариантов использования (см. рис. 4.25). Варианты использования системы заключаются в прямоугольник, снаружи которого изображаются действующие лица. Название системы может быть указано около одного из краев прямоугольника. Вариант использования обозначается эллипсом, внутри которого указывается его название. Значок «человечек» обозначает действующее лицо. Его имя ставится рядом со значком или под ним. Действующие лица соединяются с вариантами использования сплошными линиями.

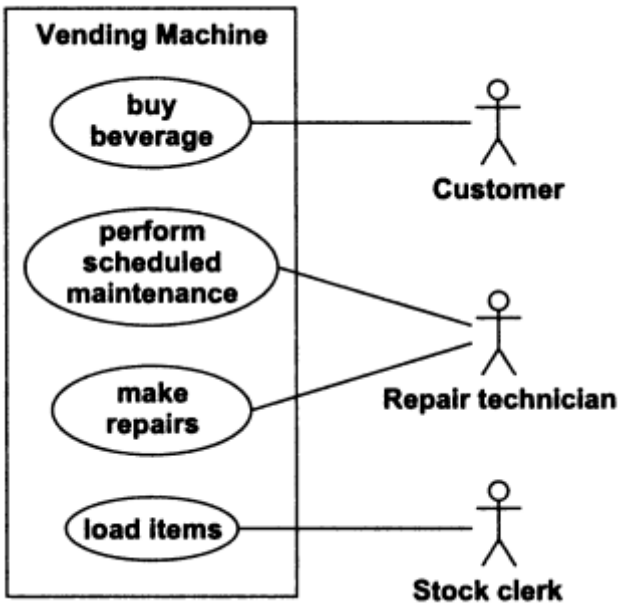

Рис. 4.25 – Варианты использования для торгового автомата

4.5.2 Модели деятельности

Диаграмма деятельности показывает последовательность этапов, образующих сложный процесс, например, вычислительный алгоритм или технологи-

ческий процесс. Диаграммы деятельности особенно полезны на ранних этапах проектирования алгоритмов и технологических процессов.

На рис. 4.26 показана диаграмма деятельности обработки заказа на покупку акций, принятого сетевой брокерской системой. Прямоугольники со скругленными углами — это виды деятельности, порядок которых обозначается стрелками. Ромбик обозначает точку принятия решения, а сплошная полоса показывает разделение или слияние параллельных потоков управления.

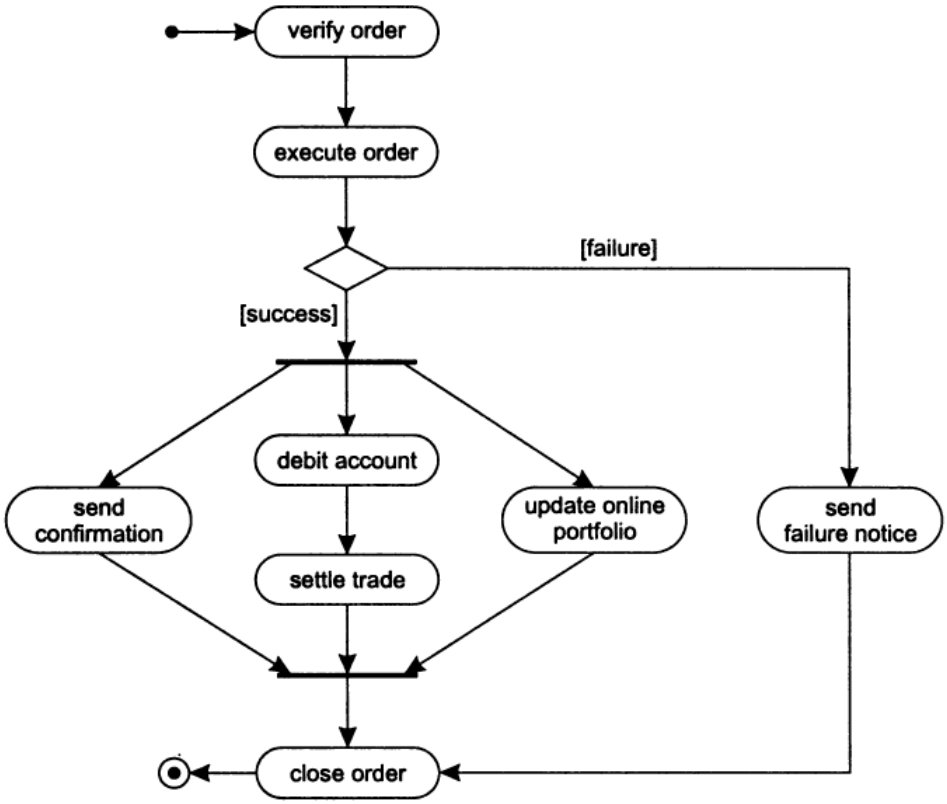

Рис. 4.26 – Диаграмма деятельности брокерской системы при обработке заказа

Сетевая брокерская система начинает с проверки заказа. Затем заказ реализуется на фондовой бирже. Если выполнение заказа проходит успешно, система выполняет три действия одновременно: отправляет клиенту подтверждение, обновляет сетевой портфель ценных бумаг с учетом результатов сделки и заканчивает сделку со второй стороной, снимая средства со счета клиента и перечисляя наличные или ценные бумаги. Когда все три параллельных потока завершаются, система объединяет их в один поток и закрывает заказ. Если же

выполнение заказа оказывается неудачным, система отправляет сообщение об этом клиенту и тоже закрывает заказ.

Диаграмма деятельности похожа на обычную блок-схему, потому что тоже показывает поток управления от этапа к этапу. Однако в отличие от блок-схемы, диаграмма деятельности может показывать одновременно параллельную и последовательную деятельность. Это отличие важно для распределенных систем. Диаграммы деятельности часто используются для моделирования организаций, потому что последние состоят из множества объектов (людей и организационных единиц), одновременно выполняющих множество операций.

Элементами диаграммы деятельности являются операции, а именно виды деятельности из модели состояний. Назначение диаграммы деятельности состоит в том, чтобы показать этапы сложного процесса и упорядочивающие ограничения, на них наложенные.

Некоторые виды деятельности выполняются до тех пор, пока не будут прерваны каким-либо внешним событием, однако в большинстве случаев деятельность имеет собственное логическое завершение. Завершение деятельности является событием завершения, которое обычно означает возможность начала выполнения следующей деятельности. Стрелка без надписи, соединяющая две деятельности на диаграмме, означает, что вторая деятельность может начаться только после того, как закончится первая.

Деятельность может быть разложена на более мелкие составляющие. На рис. 4.27 показана внутренняя структура деятельности *execute order (выполнить заказ)* с рис. 4.26. Важно, чтобы виды деятельности на одной диаграмме относились к одному и тому же уровню детализации. Сохранить уровень детализации можно, если показывать составляющие высокоуровневой деятельности на отдельных диаграммах.

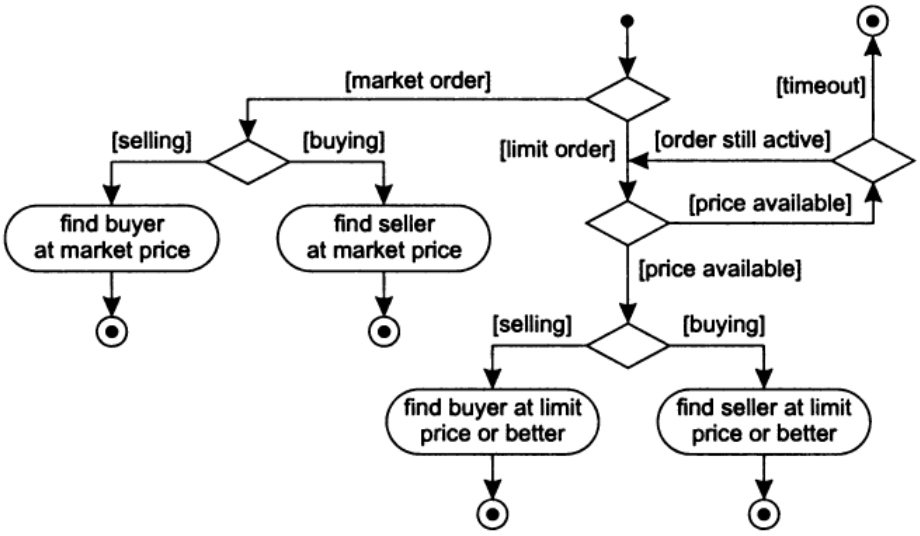

Рис. 4.27 – Диаграмма деятельности для действия *execute order*

Если какая-либо деятельность имеет несколько последующих элементов, около каждой стрелки можно указать условие в квадратных скобках, например, *[failure] ([отказ])*. Условия проверяются после завершения предшествующей деятельности. Если одно условие оказывается выполненным, стрелка, около которой оно стоит, указывает на деятельность, подлежащую выполнению. Если не удовлетворено ни одно из условий, модель можно считать плохо согласованной, потому что система зависнет, если только не получит прерывание на более высоком уровне. Во избежание подобных ситуаций следует использовать условие *[else] ([иначе]).* Если выполнено несколько условий, запущена будет только одна деятельность, причем невозможно определить, какая именно. В некоторых случаях подобная неопределенность закладывается осознанно, но часто она указывает на ошибку разработчика, который должен был учесть возможность перекрытия условий.

Для удобства в качестве обозначения множественного ветвления используется ромб с расходящимися от него стрелками, однако смысл его тот же, что и у стрелок, расходящихся от предшествующего состояния. На рис. 4.26 такой ромб имеет одну входящую стрелку и две исходящие, каждая из которых снабжена условием. При фактическом выполнении системы выбирается только одна из этих стрелок. Если несколько стрелок сходятся у некоторой деятельности,

альтернативные пути выполнения сливаются. Здесь снова можно использовать ромб со сходящимися к нему стрелками.

Сплошной кружок с исходящей стрелкой обозначает начало выполнения диаграммы деятельности. Когда диаграмма деятельности активируется, управление передается на этот кружок, а затем переходит по исходящей из него стрелке к первым этапам деятельности. Завершение диаграммы обозначается символом «бычий глаз» (сплошной кружок с кольцом вокруг него). Этот символ не может иметь исходящих стрелок.

В отличие от традиционных блок-схем организации и компьютерные системы могут выполнять несколько видов деятельности одновременно. Скорость выполнения деятельности может меняться с течением времени. Например, за одной деятельностью может следовать другая (последовательное управление), после чего может выполниться несколько параллельных видов деятельности (разделение управления), а затем они снова объединятся в одну деятельность (слияние управления). Развилка и слияние обозначаются толстой линией с одной или несколькими входящими стрелками и одной или несколькими исходящими стрелками. Для синхронизации необходимо, чтобы управление было передано в точку слияния по всем стрелкам. На развилке управление передается всем параллельным видам деятельности одновременно.

Вопросы для самоконтроля:

1. Кто может называться действующим лицом?

2. Как называются различные взаимодействия действующих лиц с системой?

3. Являются ли сбойные ситуации частью варианта использования?

4. Что показывает диаграмма деятельности?

5. Можно ли использовать диаграмму деятельности для параллельных алгоритмов действий?

5 Методологии проектирования сложных информационных систем

5.1 Методология быстрой разработки приложений

Одним из возможных подходов к разработке ИС в рамках спиральной модели ЖЦ является получившая в последнее время широкое распространение методология быстрой разработки приложений RAD (Rapid Application Development) [6]. Под этим термином обычно понимается процесс разработки ИС, содержащий три элемента:

– небольшую команду программистов (от 2 до 10 человек);

– короткий, но тщательно проработанный производственный график (от 2 до 6 мес.);

– повторяющийся цикл, при котором разработчики, по мере того, как приложение начинает обретать форму, запрашивают и реализуют в продукте требования, полученные через взаимодействие с заказчиком.

Команда разработчиков должна представлять из себя группу профессионалов, имеющих опыт в анализе, проектировании, генерации кода и тестировании программного обеспечения с использованием CASE-средств. Члены коллектива должны также уметь трансформировать в рабочие прототипы предложения конечных пользователей.

Жизненный цикл ИС методологии RAD состоит из четырех фаз: фаза анализа и планирования требований; фаза проектирования; фаза реализации; фаза внедрения.

На фазе анализа и планирования требований пользователи системы определяют функции, которые она должна выполнять, выделяют наиболее приоритетные из них, требующие проработки в первую очередь, описывают информационные потребности. Определение требований выполняется в основном силами пользователей под руководством специалистов-разработчиков. Ограничивается масштаб проекта, определяются временные рамки для каждой из последующих фаз. Кроме того, определяется сама возможность реализации данного

проекта в установленных рамках финансирования, на данных аппаратных средствах и т.п. Результатом данной фазы должны быть список и приоритетность функций будущей ИС, предварительные функциональные и информационные модели ИС.

На фазе проектирования часть пользователей принимает участие в техническом проектировании системы под руководством специалистов-разработчиков. CASE-средства используются для быстрого получения работающих прототипов приложений. Пользователи, непосредственно взаимодействуя с ними, уточняют и дополняют требования к системе, которые не были выявлены на предыдущей фазе. Более подробно рассматриваются процессы системы. Анализируется и, при необходимости, корректируется функциональная модель. Каждый процесс рассматривается детально. При необходимости для каждого элементарного процесса создается частичный прототип: экран, диалог, отчет, устраняющий неясности или неоднозначности. Определяются требования разграничения доступа к данным. На этой же фазе происходит определение набора необходимой документации.

После детального определения состава процессов оценивается количество функциональных элементов разрабатываемой системы и принимается решение о разделении ИС на подсистемы, поддающиеся реализации одной командой разработчиков за приемлемое для RAD-проектов время – порядка 60-90 дней. С использованием CASE-средств проект распределяется между различными командами (делится функциональная модель). Результатом данной фазы должны быть:

1) общая информационная модель системы;

2) функциональные модели системы в целом и подсистем, реализуемых отдельными командами разработчиков;

3) точно определенные с помощью CASE-средства интерфейсы между автономно разрабатываемыми подсистемами;

4) построенные прототипы экранов, отчетов, диалогов.

Все модели и прототипы должны быть получены с применением тех CASE-средств, которые будут использоваться в дальнейшем при построении системы. Данное требование вызвано тем, что в традиционном подходе при передаче информации о проекте с этапа на этап может произойти фактически неконтролируемое искажение данных. Применение единой среды хранения информации о проекте позволяет избежать этой ошибки.

На фазе реализации выполняется непосредственно разработка приложения, т.е. разработчики производят итеративное построение реальной системы на основе полученных в предыдущей фазе моделей, а также требований нефункционального характера. Программный код частично формируется при помощи автоматических генераторов, получающих информацию непосредственно из репозитория CASE-средств. Конечные пользователи на этой фазе оценивают получаемые результаты и вносят коррективы, если в процессе разработки система перестает удовлетворять определенным ранее требованиям. Тестирование системы осуществляется непосредственно в процессе разработки.

После окончания работ каждой отдельной команды разработчиков производится постепенная интеграция данной части системы с остальными, формируется полный программный код, выполняется тестирование совместной работы данной части приложения с остальными, а затем тестирование системы в целом. В итоге данной фазы:

1) определяется необходимость распределения данных;

2) производится анализ использования данных;

3) производится физическое заполнение базы данных;

4) определяются окончательные требования к аппаратным ресурсам;

5) определяются способы увеличения производительности;

6) завершается разработка документации проекта.

Результатом фазы является готовая система, удовлетворяющая всем согласованным требованиям.

На фазе внедрения производится обучение пользователей, организационные изменения и параллельно с внедрением новой системы осуществляется ра-

бота с существующей системой (до полного внедрения новой). Так как фаза реализации достаточно непродолжительна, планирование и подготовка к внедрению должны начинаться заранее, как правило, на этапе проектирования системы.

Следует, однако, отметить, что методология RAD, как и любая другая, не может претендовать на универсальность, она хороша в первую очередь для относительно небольших проектов, разрабатываемых для конкретного заказчика.

Методология RAD неприменима для построения сложных расчетных программ, операционных систем или других систем, требующих написания большого объема (сотни тысяч строк) уникального кода. Не подходят для разработки по методологии RAD приложения, в которых отсутствует ярко выраженная интерфейсная часть, наглядно определяющая логику работы системы (например, приложения реального времени) и приложения, от которых зависит безопасность людей (например, управление атомной электростанцией), так как итеративный подход предполагает, что первые несколько версий наверняка не будут полностью работоспособны, что в данном случае исключается.

5.2 Методология DATARUN

Современные методологии и реализующие их технологии поставляются в электронном виде вместе с CASE-средствами и включают библиотеки процессов, шаблонов, методов, моделей и других компонент, предназначенных для построения ИС того класса систем, на который ориентирована методология. Электронные методологии включают также средства, которые должны обеспечивать их адаптацию для конкретных пользователей и развитие методологии по результатам выполнения конкретных проектов.

Процесс адаптации заключается в удалении ненужных процессов, действий жизненного цикла и других компонентов методологии, в изменении неподходящих или в добавлении собственных процессов и действий, а также методов, моделей, стандартов и руководств. Настройка методологии может осу-

ществляться также по следующим аспектам: этапы и операции ЖЦ, участники проекта, используемые модели ЖЦ, поддерживаемые концепции и др.

Электронные методологии и технологии (и поддерживающие их CASE-средства) составляют ядро комплекса согласованных инструментальных средств среды разработки ИС.

Одной из наиболее распространенных в мире электронных методологий является методология DATARUN. В соответствии с методологией DATARUN ЖЦ ИС разбивается на стадии, которые связываются с результатами выполнения основных процессов, определяемых стандартом ISO 12207. Каждую стадию кроме ее результатов должен завершать план работ на следующую стадию.

Стадия формирования требований и планирования включает в себя действия по определению начальных оценок объема и стоимости проекта (условно назовем эти действия бизнес-планом ИС). Должны быть сформулированы требования и экономическое обоснование для разработки ИС, функциональная модель и исходная концептуальная модель данных, которые дают основу для оценки технической реализуемости проекта. Основными результатами этой стадии должны быть модели деятельности организации, требования к системе, включая требования по сопряжению с существующими ИС.

Стадия концептуального проектирования начинается с детального анализа первичных данных и уточнения концептуальной модели данных, после чего проектируется архитектура системы. Архитектура включает в себя разделение концептуальной модели на обозримые подмодели. Оценивается возможность использования существующих ИС и выбирается соответствующий метод их преобразования. После построения проекта уточняется исходный бизнес-план. Выходными компонентами этой стадии являются концептуальная модель данных, модель архитектуры системы и уточненный бизнес-план.

На стадии спецификации приложений продолжается процесс создания и детализации проекта. Концептуальная модель данных преобразуется в реляционную модель данных. Определяется структура приложения, необходимые интерфейсы приложения в виде экранов, отчетов и пакетных процессов вместе с

логикой их вызова. Модель данных уточняется бизнес-правилами и методами для каждой таблицы. В конце этой стадии принимается окончательное решение о способе реализации приложений. По результатам стадии должен быть построен проект ИС, включающий модели архитектуры ИС, данных, функций, интерфейсов (с внешними системами и с пользователями), требований к разрабатываемым приложениям (модели данных, интерфейсов и функций), требований к доработкам существующих ИС, требований к интеграции приложений, а также сформирован окончательный план создания ИС.

На стадии разработки, интеграции и тестирования должна быть создана тестовая база данных, частные и комплексные тесты. Проводится разработка прототипа и тестирование баз данных и приложений в соответствии с проектом. Отлаживаются интерфейсы с существующими системами. Описывается конфигурация текущей версии ПО. На основе результатов тестирования проводится оптимизация базы данных и приложений. Приложения интегрируются в систему, проводится тестирование приложений в составе системы и испытания системы. Основными результатами стадии являются готовые приложения, проверенные в составе системы на комплексных тестах, текущее описание конфигурации ПО, скорректированная по результатам испытаний версия системы и эксплуатационная документация на систему.

Стадия внедрения включает в себя действия по установке и внедрению баз данных и приложений. Основными результатами стадии должны быть готовая к эксплуатации и перенесенная на программно-аппаратную платформу заказчика версия системы, документация сопровождения и акт приемочных испытаний по результатам опытной эксплуатации.

Стадии сопровождения и развития включают процессы и операции, связанные с регистрацией, диагностикой и локализацией ошибок, внесением изменений и тестированием, проведением доработок, тиражированием и распространением новых версий ПО в места его эксплуатации, переносом приложений на новую платформу и масштабированием системы.

Стадия развития фактически является повторной итерацией стадии разработки.

Методология DATARUN базируется на системном подходе к описанию деятельности организации. Построение моделей начинается с описания процессов, из которых затем извлекаются первичные данные (стабильное подмножество данных, которые организация должна использовать для своей деятельности). Первичные данные описывают продукты или услуги организации, выполняемые операции (транзакции) и потребляемые ресурсы. К первичным относятся данные, которые описывают внешние и внутренние сущности, такие как служащие, клиенты или агентства, а также данные, полученные в результате принятия решений, как например, графики работ, цены на продукты.

Основной принцип DATARUN заключается в том, что первичные данные, если они должным образом организованы в модель данных, становятся основой для проектирования архитектуры ИС. Архитектура ИС будет более стабильной, если она основана на первичных данных, тесно связанных с основными деловыми операциями, определяющими природу бизнеса.

Любая ИС представляет собой набор модулей, исполняемых процессорами и взаимодействующих с базами данных. Базы данных и процессоры могут располагаться централизованно или быть распределенными. События в системе могут инициироваться внешними сущностями (такими как клиенты у банкоматов) или временные события (конец месяца или квартала). Все транзакции осуществляются через объекты или модули интерфейса, которые взаимодействуют с одной или более базами данных.

Подход DATARUN преследует две цели:

1) определить стабильную структуру, на основе которой будет строиться ИС. Такой структурой является модель данных, полученная из первичных данных, представляющих фундаментальные процессы организации;

2) спроектировать ИС на основании модели данных.

Объекты, формируемые на основании модели данных, являются объектами базы данных, обычно размещаемыми на серверах в среде клиент/сервер. Объек-

ты интерфейса, определенные в архитектуре компьютерной системы, обычно размещаются на клиентской части. Модель данных, являющаяся основой для спецификации совместно используемых объектов базы данных и различных объектов интерфейса, обеспечивает сопровождаемость ИС.

Вопросы для самоконтроля:

1. В чем заключаются основные принципы методологии быстрой разработки приложений?

2. Какие основные фазы имеются в жизненном цикле разработки информационной системы по методологии быстрой разработки приложений?

3. Какие недостатки имеются у методологии быстрой разработки приложений?

4. Какие отличия есть у методологии DATARUN по сравнению с методологией быстрой разработки приложений?

5. Какие основные стадии имеются в жизненном цикле разработки информационной системы по методологии DATARUN?

Глоссарий

SADT-методология (Structured Analysis and Design Technique) – совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области.

UML (Unified Modeling Language) – стандарт обозначений для объектно-ориентированного моделирования.

Базовый элемент системы – элементы, описания которых являются известными и дальнейшему делению не подлежат.

Восходящее проектирование – процесс проектирования, при котором решение задач низких иерархических уровней предшествует решению задач высоких иерархических уровней.

Жизненный цикл информационной системы – это совокупность стадий и этапов, которые проходит ИС в своем развитии от момента принятия решения о создании системы до момента прекращения функционирования системы.

Информационная система (ИС) – это организационно-техническая система, использующая информационные технологии в целях обучения, информационно-аналитического обеспечения человеческой деятельности и процессов управления.

Количество информации – это мера уменьшения энтропии объекта после совершения некоторого события.

Математическая модель технического объекта – система математических объектов (чисел, переменных, матриц, множеств и т.п.) и отношений между ними, отражающих некоторые свойства технического объекта.

Нисходящее проектирование – процесс проектирования, при котором решение задач высоких иерархических уровней предшествует решению задач более низких иерархических уровней.

Один бит – это количество информации, получаемое при осуществлении одного из двух равновероятных событий.

Параметр модели – это количественное выражение некоторого свойства характеристики модели.

Подсистема – это относительно независимая часть системы, которая обладает всеми свойствами системы и, в частности, имеет свою подцель, на достижение которой эта подсистема и ориентирована.

Проект информационной системы – это проектно-конструкторская и технологическая документацию, в которой представлено описание проектных решений по созданию и эксплуатации ИС в конкретной программно-технической среде.

Проектирование информационной системы – это процесс преобразования входной информации об объекте проектирования, о методах проектирования и об опыте проектирования объектов аналогичного назначения в соответствии со стандартами в проект ИС.

Сигнал (в информационном смысле) – материальный носитель информации, т.е. средство переноса информации в пространстве и во времени.

Система – это конечное множество функциональных элементов и отношений между ними, выделяемое из среды в соответствии с определенной целью в рамках определенного временного интервала.

Сообщение – совокупность знаков или периодических сигналов, содержащих информацию.

Технология проектирования – это совокупность методологии и средств проектирования ИС, а также методов и средств организации проектирования.

Типовая проектная процедура – процедура, предназначенная для многократного применения при проектировании многих типов объектов.

Унифицированные (стандартные) элементы – это элементы, которые разрабатываются раньше, чем та или иная конкретная система из определенного класса.

Элемент системы – некоторый объект, обладающий рядом важных свойств и реализующий в системе определенный закон функционирования, причем, внутренняя структура данного объекта не рассматривается.

Энтропия – это мера неопределенности какого-либо опыта, который может иметь разные исходы.

Литература

1. Теория систем и системный анализ: учебное пособие для вузов / А.М. Кориков, С.Н. Павлов – Томск: ТУСУР, 2007. – 343 с.

2. Павлов С.Н. Теория систем и системный анализ: Учебное пособие. – Томск: Томский межвузовский центр дистанционного образования, 2003. – 134 с.

3. Тимаков С.О. Информационные системы в социальной работе: Учебно-методическое пособие. – Томск: ТУСУР, 2003. – 132 с.

4. Адуева Т.В. Автоматизированный бухгалтерский учет и основы аудита: Учебное пособие. – Томск: Томский межвузовский центр дистанционного образования, 2003. – 189 с.

5. Смирнова Г.Н., Сорокин А.А., Тельнов Ю.Ф. Проектирование экономических информационных систем: Учебник. – М.: Финансы и статистика, 2002. – 512 с.: ил.

6. Золотов С.Ю. Основы проектирования информационных систем: Учебное пособие / Томск: ТУСУР, 2007. – 68 с.

7. Дж. Рамбо, М. Блаха. UML 2.0. Объектно-ориентированное моделирование и разработка. 2-е изд. – СПб: Питер, 2007. – 544 с.: ил.

8. Описание стандартов семейства IDEF [Электронный ресурс]. – Режим доступа: http://idef.ru/, свободный.

9. Object Management Group [Электронный ресурс]. – Режим доступа: http://omg.org/, свободный.