Федеральное агентство по образованию

АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Серия «Учебно-методический комплекс дисциплины»

А.А. Кудинов

Проектирование автоматизированных систем

Указания к практическим занятиям по объектно-ориентированным методам проектирования

Учебное пособие

Благовещенск
2010

Печатается по решению
редакционно-издательского совета
энергетического факультета
Амурского государственного
университета

А.А. Кудинов Проектирование автоматизированных систем. Указания к практическим занятиям по объектно-ориентированным методам проектирования. Учебное пособие. – Благовещенск: Амурский гос. ун-т, 2010.

Пособие предназначено для студентов специальности 220301 «Автоматизация технологических процессов и производств», изучающих дисциплину «Проектирование автоматизируемых систем» и выполняющих практические работы по дисциплине. Может быть использовано при выполнении курсовых и дипломных проектов.

Рецензенты: А.Н. РыбалевА.В. заведующий кафедрой автоматизации производственных процессов и электротехники, канд. техн. наук, доцент;
Кушниренко Г.И., доцент кафедры электропривода и автоматизации АПК и электрооборудования тракторов и автомобилей ДальГАУ.

В авторской редакции

ПРЕДИСЛОВИЕ

В отличие от традиционных методов проектирования, изложенных в работе [8] процесс проектирования автоматизированных систем управления нацелен на объект. Объектно-ориентированные методы проектирования систем управления можно представить как систему взаимодействия следующих уровней обеспечения:

концептуального;

информационного;

функционального;

формализации;

алгоритмического;

программного;

аппаратного.

В настоящей работе даются методические указания по изучению некоторой части этой системы: процесс управления как последовательность операций, циклограмма, алгоритм, сеть Петри, дерево достижимости, основные компоненты языка UML с концептуальным проектированием реляционных баз данных. Изучение этих рекомендаций с разработкой самостоятельных концепций и решений позволит студенту освоить нетрадиционные методы проектирования систем автоматизированного управления различными объектами.

Работа 1

ПРОЦЕСС УПРАВЛЕНИЯ КАК ПОСЛЕДОВАТЕЛЬНОСТЬ ОПЕРАЦИЙ

В качестве примера рассмотрим последовательность операций управления лифтовым механизмом. Структура одного из возможных вариантов такой последовательности операций представлена на рисунке 1. Роль определителей текущего состояния лифтовой системы будут выполнять датчики параметров и положений лифта при его работе. На схеме (рисунок 1) отдельные операции и этапы отображены в виде блоков, пронумерованными порядковыми номерами. Работа лифта начинается с команды «Пуск» - блок 1. Блок 2 проверяет условие: это первичный пуск или повторный. Если это повторный пуск, то предполагается, что это сбой в работе, аварийная ситуация (блок 3), после выявления, которой необходимо остановит лифт (блок 4). При срабатывании первичного пуска должно произойти включение привода движения кабины лифта – блок 5. Факт включения привода контролируется блоком 6. Если привод не сработал, то это определяется как аварийная ситуация и управление передается блоку 3 (аварийная ситуация). Если же привод включился, то вступает в работу схема идентификации номера этажа, на который следует подняться (или спуститься) лифтовой кабине. Блоком 7 контролируется факт достижения лифтом нужного этажа. Если кабина достигла заданного этажа, то производится останов привода движения кабины – блок 8 и проверяется – сработала ли команда останова кабины – блок 9. Если кабина не остановилась, то это фиксируется как аварийная ситуация и идет обращение к блоку 3. Если же кабина остановилась, то, через определенный промежуток времени, определяемый блоком 10, включается механизм открывания дверей кабины – блок 11, с проверкой факта открытия двери – блок 12. Если дверь открылась, то через заданный промежуток времени, задаваемый блоком 13, процесс управления возвращается автоматически на продолжение работы –

блок 1. В том случае если дверь кабины не открылась, то через определенный промежуток времени, который задается блоком 14, повторно проверяется факт открытия двери и, в случае повторного срабатывания двери, процесс также возвращается к начальному блоку – блоку 1. Если же и при повторной попытке дверь не открылась, то блок 15 обращается к блоку аварийной ситуации – блоку 3. Описанная последовательность операций при работе лифта лишь фрагмент процесса управления лифтовой системой – здесь отсутствуют такие вопросы как:

выполнение стратегии приоритетов при одновременном вызове кабины с разных этажей;

анализ ситуации останова кабины между этажами.

Основной целью примера является демонстрация основных принципов управления конкретным объектом, а не практическая реализация. Конкретная практическая реализация подобной системы управления потребует:

технического анализа процессов управления, определение спецификаций и разработки технического задания;

разработки технического предложения;

разработки моделей функционирования с учетом ограничений и параметров;

проведения расчетных операций и конструкторских разработок;

проработки алгоритмов процессов;

разработки прикладного программного обеспечения;

параллельной разработки аппаратных средств реализации и интерфейсов связи и т.п.;

выпуска полной проектной и технической документации по разработке системы на всех этапах проектирования.

Домашнее задание. Разработать процесс управления как последовательность операций для конкретной системы управления реальным объектом, выявленным студентом на производственной практике для курсового и дипломного проектирования. Процесс управления должен быть представлен в

диаграмме последовательности операций. При отсутствии реального объекта, преподаватель выдает учебное задание, для которого составляется диаграмма. Выполненная схема должна отвечать требованиям ЕСКД и стандарту АмГУ.

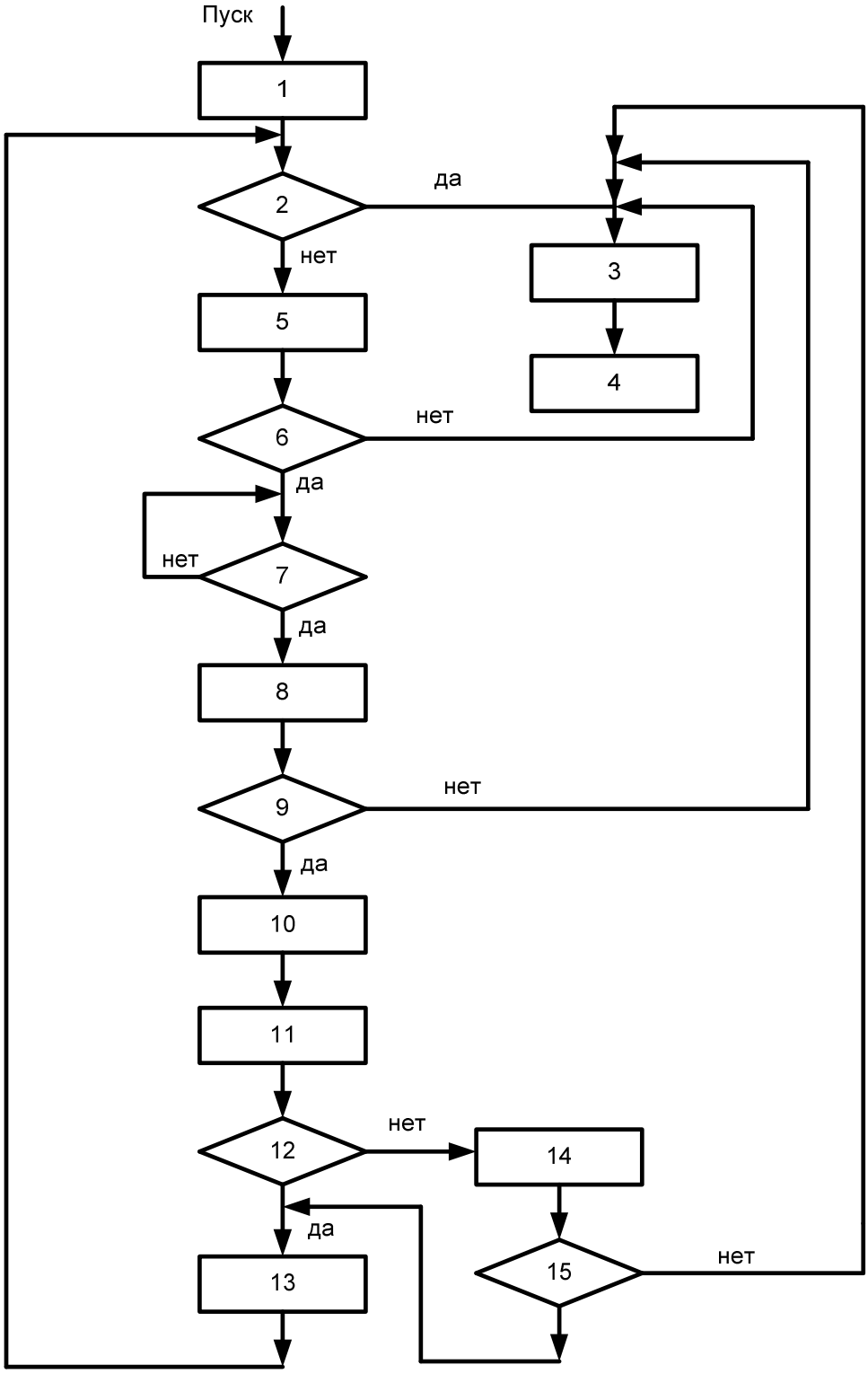

Рисунок 1 – Диаграмма последовательности операций при работе лифтовой системы

Работа 2
ЦИКЛОГРАММА

На рисунке 2 представлена циклограмма фрагмента функционирования лифтовой системы, разработанная на основе перечня операций, описанных выше.

В циклограмме на рисунке 2 сохраняются те же операции и обозначения элементов управления и контроля, что и в модели, изображенной на рисунке 1, но без детализации и описания технических параметров этих элементов, поскольку на этом этапе важны лишь функциональные, внешние параметры процедуры управления. Временные параметры работы представлены так же чисто качественно, в виде условных циклов времени, без привязки к конкретной продолжительности этих циклов. Каждый цикл привязан либо к операции управления, либо операции контроля и графически обозначен в виде отрезка прямой линии. Последовательные функциональные связи операций изображены на циклограмме в виде дуг со стрелками, расположенными в нижней части цепочки циклов. Дуги, расположенные сверху цепочки циклов, показывают функциональные связи в различных вариантах неправильной работы или аварийных ситуациях. Резюмируя проведенный анализ функциональности представления процессов управления объектами можно отметить следующие характерные особенности:

типичными и широко применяемыми в практике прикладными процессами, которые оснащаются системами управления, являются циклические процессы;

средства отображения функциональности процесса управления объектами дают лишь качественную оценку процессам управления, и это обстоятельство доказывается, в частности, на примере использования циклограмм;

необходимость расширения представления функциональных свойств процесса управления требует применения более эффективного аппарата анализа и синтеза – аппарата моделирования процессов управления;

рассмотрение функциональных характеристик системы управления является методологической основой для следующего этапа – этапа моделирования процессов управления.

Домашнее задание. На основе предыдущего домашнего задания разработать циклограмму управления и контроля реального объекта. При этом количество операций и элементов управления может быть больше или меньше, чем в построенной циклограмме (рисунок 2). Циклограмма функционирования реального объекта строиться для разных индивидуальных задач в реальном времени. Следует дать пояснения к своим циклограммам и выполнить в соответствии с требованиями ЕСКД и стандарта АмГУ.

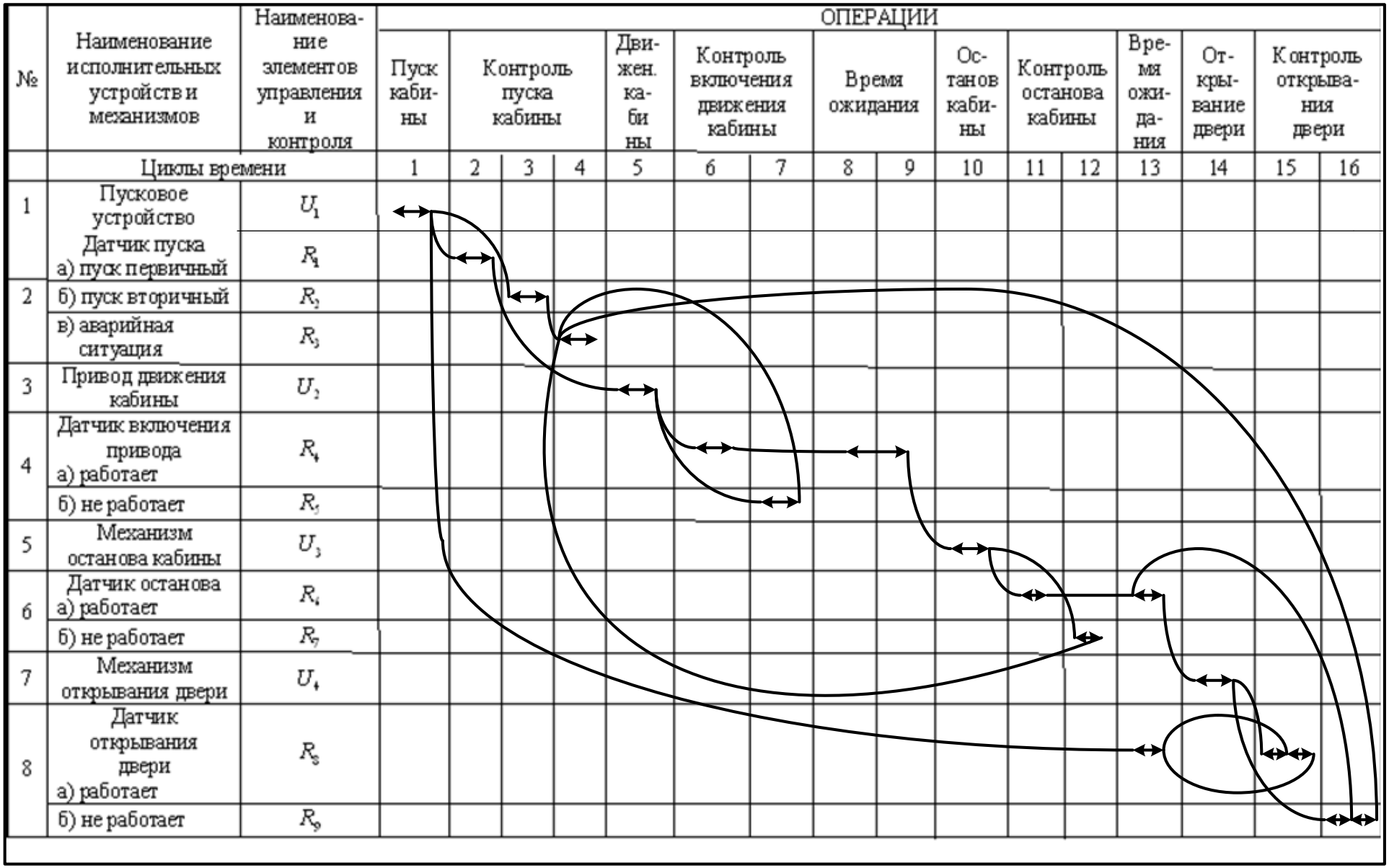

Рисунок 2 – Циклограмма функционирования лифтовой системы

Работа 3

АЛГОРИТМ ЛИФТА

Вернемся к примеру управления лифтовой системой, который был описан в работах 1 и 2, где процесс управления был представлен в виде циклограммы. Циклограмма (рисунок 2) наглядно демонстрирует общие принципы функционирования лифтовой системы под действием сигналов управления, показывает связи между отдельными операциями, но она весьма неудобна для следующего этапа алгоритмизации процесса управления. Поэтому необходимо применить такую форму представления модели управления, которая устраняла бы указанный недостаток. Представим процесс управления известной нам по предыдущим описаниям лифтовой системы, моделью в виде блок-схемы, изображенной на рисунке 3. Ранее был рассмотрен этот же пример проектирования процесса управления лифтом на концептуальном уровне, в виде описания последовательности операций, с применением того же метода блок-схем (рисунок 1). На блок-схеме рисунка 1 каждый блок представлял лишь порядковый номер очередной операции управления или контроля управляющих действий системы, а содержательная часть каждой операции требовала отдельного, текстового описания процесса управления лифтом.

На этапе моделирования можно применить тот же метод формального описания процесса управления, но уже с кратким описанием каждой операции внутри блоков, что и демонстрирует блок-схема на рисунке 3.

Описанный фрагмент процесса управления лифтовой системы описывает лишь часть работы системы и сюда не включены схемы приоритетов при одновременном запросе разных этажей, ситуации останова кабины лифта между этажами, аварийные вызовы и т.п.

Основная цель данной модели чисто методическая и иллюстративная: показать, каким-образом можно формализовать весьма доступными методами процесс функционирования или проектирования системы управления конкретным объектом.

Реализация этих моделей может быть выполнена на разных уровнях:

на аппаратном уровне в виде электронной схемы, когда все команды управления и операции контроля «прошиты» на схемном уровне;

на программно-аппаратном уровне, когда схемы соединений, силовые элементы, физический уровень реализации сигналов представлен в виде электронных схем, а сама логика управления, последовательность команд реализована в виде компьютерной программы. Второе решение применяется наиболее часто, поскольку оно более гибко и позволяет реализовать на одном объекте различные способы и методы процессов управления, разрабатывая лишь новые варианты моделей и их программную реализацию, а, не прибегая к перестройке электронных схем, что значительно сложнее в реализации.

Программный уровень определяет конкретные команды, которые реализуются на аппаратном уровне в виде последовательности сигналов, подаваемых на следующие типовые элементы исполнения:

элементы включения приводов движения (например, включение привода движения кабины лифта) с помощью пускателей, реле, контакторов и т.п.;

элементы контроля и анализа правильности исполнения команд управления – датчики различных типов;

элементы оповещения и сигнализации – экраны, табло;

элементы поддержания диалога и других средств интерфейса пользователя – клавиатуры, мониторы, мыши и т.п.

В любом случае после рассмотрения обобщенных моделей управления, подобной рассмотренной выше модели управления лифтовой системой (рисунок 3), мы должны определить конкретные элементы управления и контроля, которые необходимо применить при реализации системы управления. Для этого проводится подробный анализ функционирования всех этапов управления, и назначаются конкретные элементы управления и контроля (пускатели, реле, выключатели, датчики) на всех уровнях управления объектом.

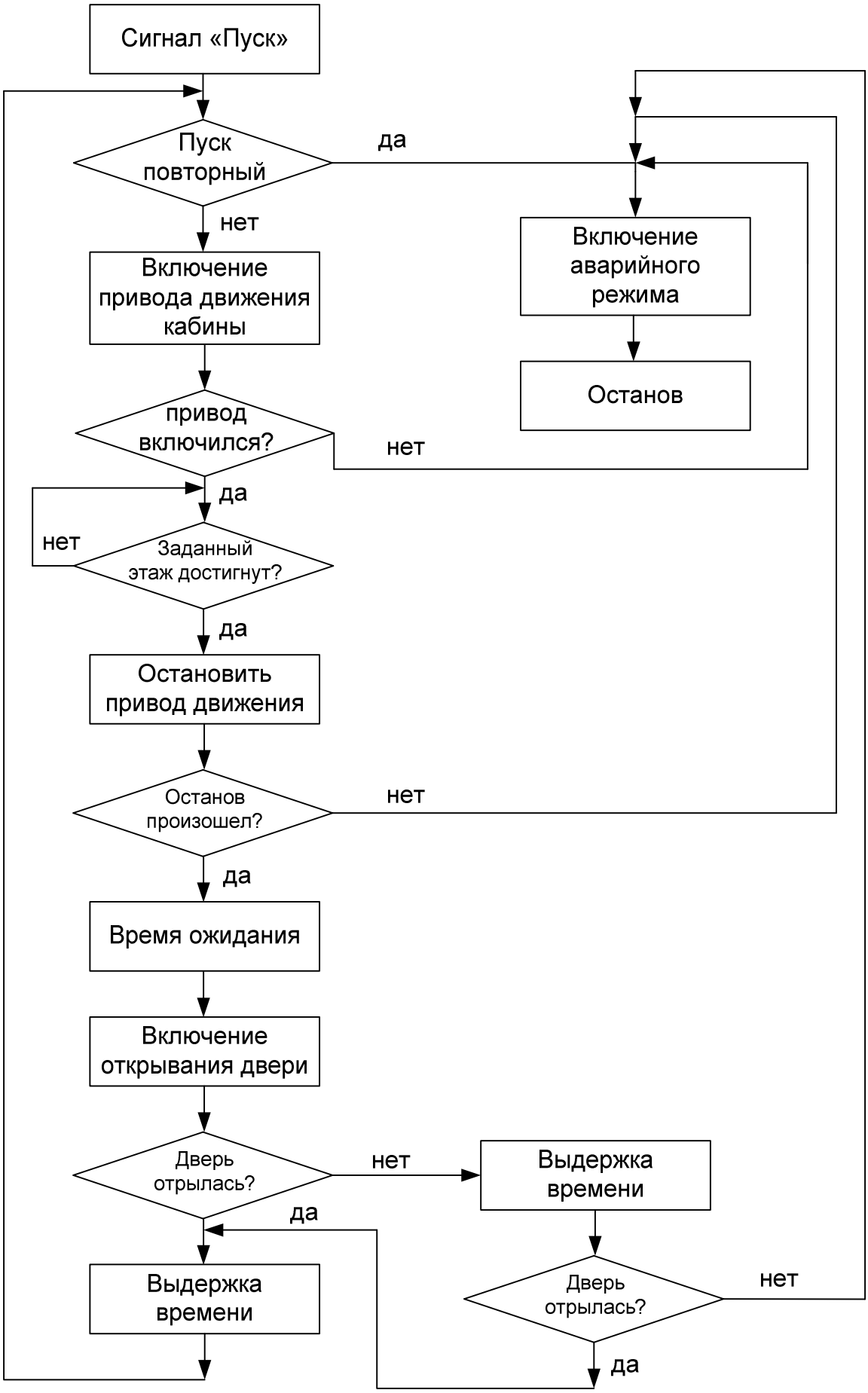

Рисунок 3 – Схема алгоритма функционирования лифтовой системы

Обычно все эти элементы в виде перечня удобнее свести в единую таблицу, назначив элементам имена и обозначения. Процесс управления непосредственно объектом в реальном времени включает два основных, взаимосвязанных информационных потока: поток управляющих воздействий и поток осведомительных сигналов, который контролирует текущее состояние процесса управления. Поэтому, прежде всего, необходимо определить и назначить те функциональные элементы, которые воспринимают и реализуют эти два потока (работа 2).

Анализ последовательности операций изображенных в модели на рисунке 3 показывает, что:

модель имеет замкнутый характер, с отслеживанием обратных связей и законченными рабочими циклами;

модель функционально подготовлена для разработки следующего этапа – алгоритмизации процессов управления.

И, тем не менее, модель процесса управления лифтовой системой, представленная на рисунке 3, не является функционально полной, а самое главное не отражает динамических процессов в управлении объектами, поскольку не содержит ответов на следующие важные вопросы:

обладает ли модель свойством живучести;

реализует ли она параллельные процессы;

обладает ли она способностью разрешать конфликтные ситуации;

имеет ли механизмы анализа достижимости заданных ситуаций.

Поэтому более функционально полными являются модели процессов управления, которые способны ответить на сформулированные выше вопросы. Такими моделями являются сети Петри.

Домашнее задание. Рекомендуется на основе предыдущих работ построить схему алгоритма функционирования реального объекта.

Работа 4

СЕТЬ ПЕТРИ

Аппарат сетей Петри достаточно широко известен и описан во многих источниках, поэтому здесь будут приведены краткие ссылки на некоторые положения этого аппарата.

В сети Петри представлены состояния объектов, которые отображаются позициями сети и действия, которые отображаются переходами. Позиции графически представляются в виде кружочков, а переходы в виде планок. Позиции и переходы связаны между собой направленными дугами, которые обозначают функциональную связь между ними. Простое изображение совокупности позиций и переходов в виде сети отображает лишь статическую компоненту сети. Для определения динамической составляющей применяется правило маркировки сети. Суть ее в том, что вводятся специальные правила срабатывания переходов при соблюдении определенных условий.

Сеть Петри, построенная для нашего примера – описания процесса управления лифтовой системой – изображена на рисунке 4.

В графе сети Петри, изображенном на рисунке 4, приняты следующие обозначения позиций, переходов и блоков:

Позиции:

P_1 - начальное состояние кабины лифта;

P_2 - состояние первичного пуска кабины;

P_3 - состояние вторичного пуска кабины;

P_4 - аварийное состояние;

P_5 - анализ включения привода движения кабины;

P_6 - анализ достижения заданного этапа;

P_7 - анализ останова кабины;

P_8 - анализ открывание двери кабины лифта;

P_9 - повторный анализ открывания двери кабины лифта;

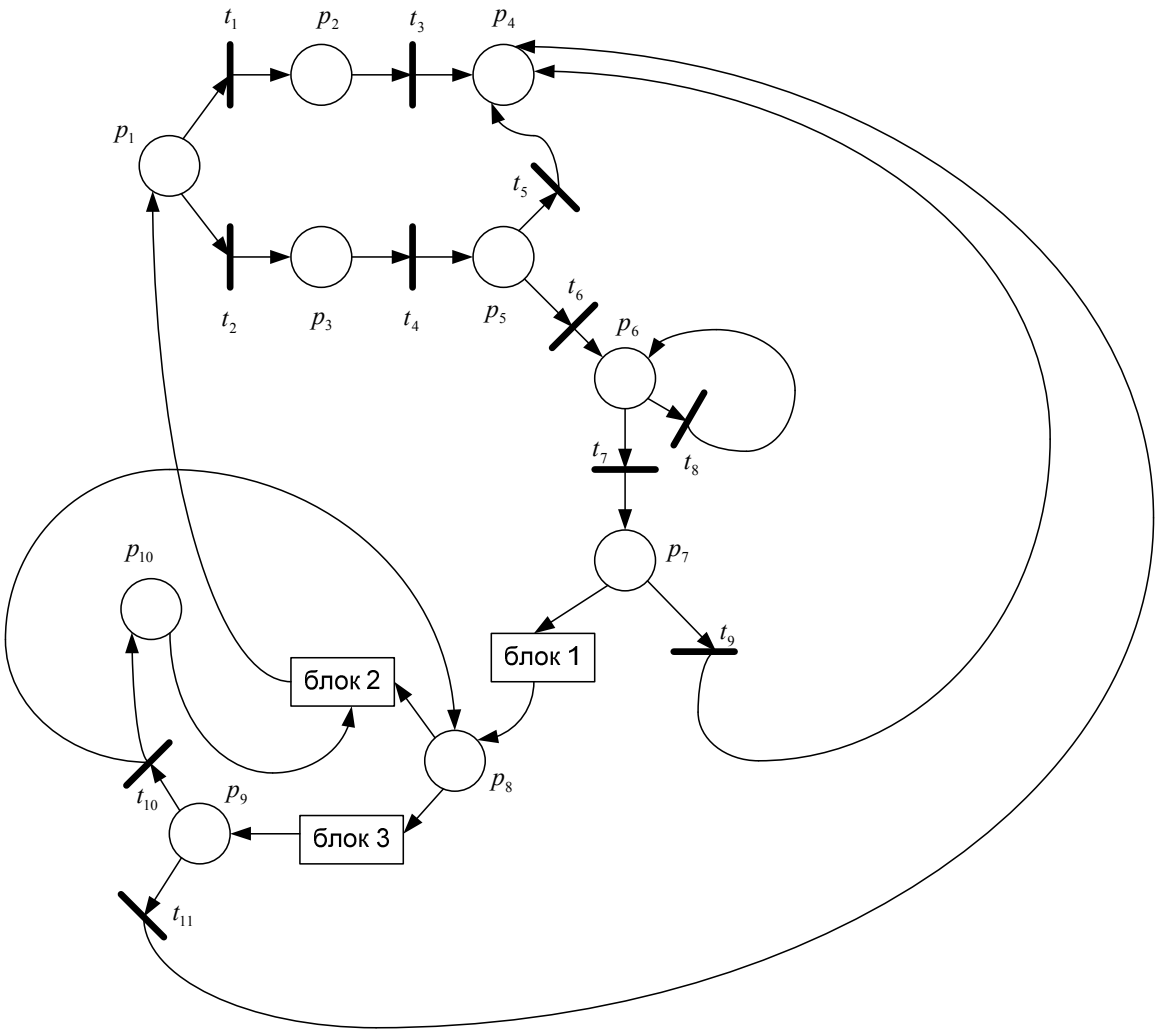

Рисунок 4 – Сеть Петри для моделирования процесса управления работой кабины лифта

P_{10} - состояние открытой двери после вторичного анализа.

Переходы:

t_1 - команда пуска кабины лифта;

t_2 - команда повторного пуска кабины;

t_3 - команда аварийного останова кабины после повторного пуска;

t_4 - команда включения привода движения кабины лифта после пуска;

t_5 - сообщение о не включении привода движения кабины;

t_6 - сообщение о включении привода движения кабины;

t_7 - команда на останов кабины;

t_8 - сообщение о текущем этаже при движении кабины;

t_9 - сообщение о невозможности достижения заданного этажа;

t_{10} - сообщение о срабатывании привода открывания двери;

t_{11} - сообщение о вторичном отказе при открывании двери.

Блоки:

блок 1 – элемент организации временного интервала, равного циклу прохождения кабиной лифта от начала пуска до останова на заданном этаже. Элемент включает также команду на открывание двери при достижении заданного этажа;

блок 2 – элемент организации временного интервала, который включает время ожидания после открытия двери кабины до очередного пуска кабины. Блок содержит также сообщение об окончании цикла реализации вызова кабины лифта;

блок 3 – элемент организации временного интервала, который включает время ожидания при повторной проверке включения механизма открывания двери кабины.

Напомним некоторые краткие положения основ теории сетей Петри.

Сеть Петри, изображенная на рисунке 4 является классической сетью, содержащей изображения состояний элементов системы управления в виде кружков, а изображения событий, которые здесь называются переходами в виде планок. Состояния обозначаются символом P_n, а переходы – символом t_m, где n и m – любые целые числа в арабской нотации, обозначающие лишь порядковые номера соответственно событий и переходов. Два соседних состояния всегда связаны как минимум одним переходом.

В конкретных приложениях состояния могут отображать как положение какого-то объекта, так и состояние анализа ситуации. В свою очередь переходы могут отображать как конкретные команды управления объектом, так и сообщения о некоторых событиях.

Третьим элементом сети является блок, который не имеет специального символьного обозначения и изображается в сети в виде прямоугольника и служит для изображения сложного перехода.

При этом обозначение блока произвольно и обычно внутри прямоугольника изображающего блок пишут, например, блок 2.

Обычный переход P_i отображает факт мгновенного изменения одного состояния на другое и при моделировании это игнорирование временных задержек часто оправдано, особенно при анализе качественных параметров процессов управления.

Но иногда, необходимо учитывать тот факт, что переход t_i из одного состояния P_i в другое состояние P_{i+1} связано с реализацией ряда промежуточных внутренних операций. Другими словами, вход в переход и выход из него представляют собой некоторое промежуточное временное преобразование, которое необходимо учитывать. Для отображения таких ситуаций и применяется элемент сети называемый блоком.

Например, в сети, изображенной на рисунке 4, блоками изображены переходы, связанные с искусственной задержкой времени, необходимой для рабочего цикла лифтовой системы. Так, для фиксирования интервала времени между остановом кабины лифта и моментом открывания двери применяется блок 1. Кроме этого временного интервала блок 1 включает и команду на включение открывания двери кабины лифта. Блок 2 в этой сети необходим для фиксирования временного интервала между моментом открывания двери и следующим пуском кабины. Блок 2 включает в себя также сообщение об окончании цикла вызова. Блок 3 фиксирует временной интервал ожидания при повторном анализе факта открывания двери. Количественное содержание временного интервала, реализуемого блоками 1, 2 и 3 определяются при разработке технических спецификаций при проектировании системы управления.

Содержание позиций и переходов, описанных выше для сети Петри, изображенной на рисунке 4, достаточно полно отображает последовательность

операций управления и контроля при управлении лифтовой системой, функциональные связи между операциями, но при этом не видно преимущества сети перед ранее представленными моделями процесса управления лифтовой системой, изображенными на рисунке 1 и рисунке 3. Сеть Петри получает преимущество перед другими моделями при введении процедуры маркировки сети.

Из теории сетей Петри известно, что маркировка представляет собой процесс присвоения фишек позициям сети. Фишка – произвольное число, условно изображаемое на графе сети темным кружком внутри позиции. Фишки используются для определения выполнения сети. Первоначальное распределение фишек в сети, как и их количество в позициях может быть любым и определяется лишь целями маркировки.

Фишки управляют выполнением переходов в сети, причем выполнением сети представляет собой запуск переходов. Переход запускается удалением фишек из его входных позиций и образованием новых фишек, помещаемых в его выходные позиции. Переход может запускаться только в том случае, если он разрушен.

Разрушенным называется переход, в котором каждая из его входных позиций имеет число фишек, по крайней мере, равное или большее числу входных дуг. При срабатывании разрешенного перехода из входной позиции этого перехода удаляется в общем случае столько фишек, сколько имеется входных дуг из позиции в переход. Это правило относится и к образованию фишек в выходной позиции перехода.

Применение процедуры маркировки в сетях Петри позволяет получить от модели, следующие дополнительные возможности:

посмотреть процесс управления в динамике;

проследить процессы реализации каждого действия (перехода) и последовательности переходов;

оценить параметры живучести и безопасности сети;

выявить конфликтные ситуации и найти способы разрешения их;

реализовать проблему разделения общего ресурса между двумя параллельными процессами;

решить проблему синхронизации параллельных процессов.

Домашнее задание. Необходимо выполнить модель реального объекта с помощью рассмотренной сети Петри и рассмотреть процесс управления в динамике.

Работа 5

ДЕРЕВО ДОСТИЖИМОСТИ

В сети представленной на рисунке 4 изображена начальная маркировка с единичной фишкой позиции P_i.

Аналитически такая маркировка может быть представлена в виде следующей записи: $m_0 = (1,0,0,0,0,0,0,0,0,0)$, где каждая цифра ряда показывает количество фишек в соответствующей по номеру позиции. Такая маркировка продиктована следующими причинами.

Анализ начального состояния сети показывает, то сеть должна быть замкнута на первичное ожидание сигнала пуска кабины лифта после завершения цикла обслуживания предыдущего запроса на передвижение. Поэтому в начальной маркировке сети присутствует одна фишка, в начальной позиции ожидания пуска – вершина P_1.

Одним из главных свойств сети Петри является возможность исследования моделируемого сетью процесса на достижимость определенных состояний. Факт достижимости определяется в процессе последовательного срабатывания разрешенных переходов. Каждое срабатывание очередного перехода вызывает изменение маркировки сети по правилам, которые были описаны выше.

Согласно этим же правилам последовательное изменение маркировок можно представить в виде диаграммы, называемой деревом достижимости. Фактически дерево достижимости – это графическая модель процесса срабатывания всех возможных вариантов последовательностей переходов. Такая модель позволяет наглядно исследовать и проанализировать все ситуации, достижение которых представляет интерес в конкретной модели приложения или предметной области.

Дерево достижимости, построенное для примера управления лифтовой системой, приведено на рисунке 5.

Начальная маркировка сети - m_0 показывает как бы стартовое положение сети, при котором фишка находится в одной позиции – в первой.

В левой части дерева сгруппированы особые позиции: позиции, характеризующие наступление аварийных ситуаций (они подчеркнуты сплошной линией) и позиции возврата в начальное состояние системы после реализации очередного пуска кабины лифта (они выделены жирным шрифтом).

При анализе ветвей дерева достижимости можно отметить несколько важных моментов:

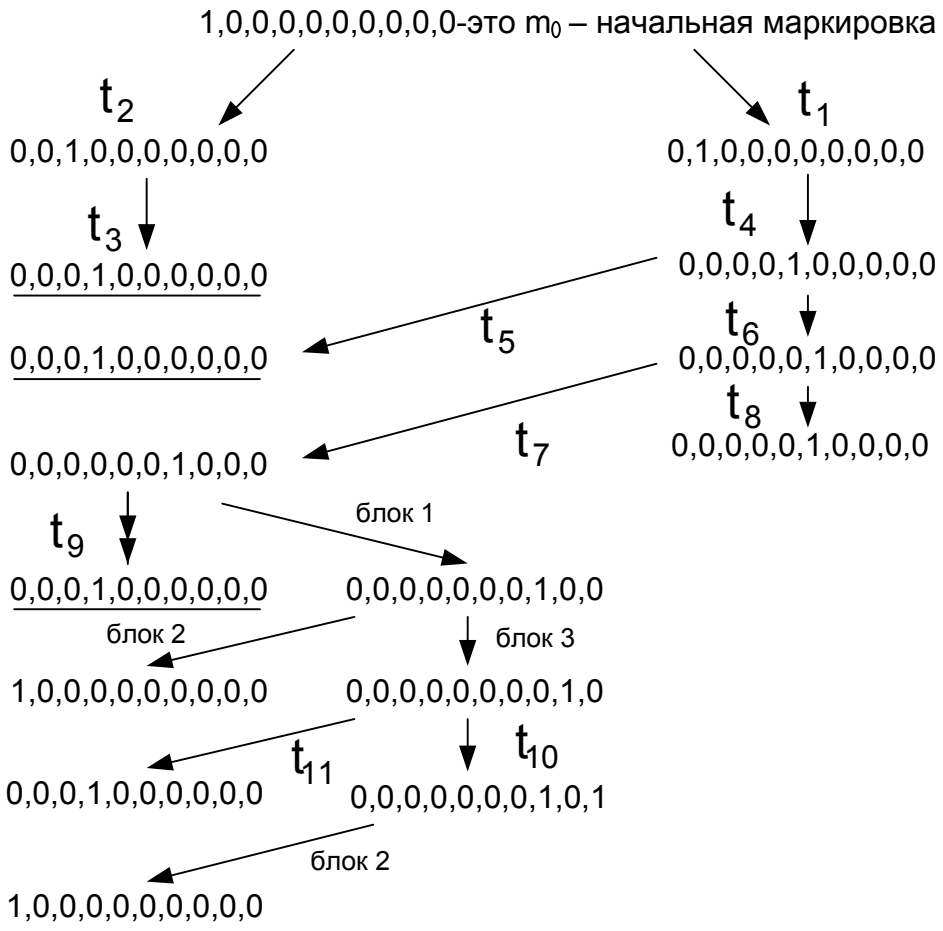

Рисунок 5 – Дерево достижимости сети Петри при моделировании лифтовой системы

1. При необходимости повторного пуска кабины лифта (переходы t_2 - t_3) достигается тупиковая ситуация, при которой не при каких условиях не может сработать не один переход в сети. По существу эта ситуация отображает аварийный режим работы лифта.

Подобные ситуации достигаются в сети также после:

срабатывания переходов t_1-t_4-t_5;

срабатывания переходов t_1-t_4-t_6-t_7-t_9;

срабатывания переходов t_1-t_4-t_6-t_7-блок 1-блок 3-t_{11}.

В первом случае это иллюстрирует тот факт, что для определения аварийной ситуации, например, по несрабатыванию пуска привода движения кабины лифта, необходимо три уровня (такта) реализации переходов.

Во втором случае, для определения подобной ситуации, но по отношению к факту отсутствия останова кабины лифта, необходимо пять тактов срабатывания переходов. В третьем случае для анализа возможной аварийной ситуации при открывании двери кабины лифта необходимо семь тактов срабатывания переходов в сети.

2. При правильной работе лифта возврат в исходное состояние – ожидание следующего пуска кабины лифта после выполнения предыдущего вызова – достигается после выполнения всех необходимых действий, составляющих последовательность операций автоматического цикла работы лифтовой системы.

Это происходит либо за шесть тактов (переходы t_1-t_4-t_6-t_7-блок 1-блок 2), либо за восемь тактов, с учетом повторной попытки открывания двери кабины лифта (переходы t_1-t_4-t_6-t_7-блок 1-блок 3-t_{10}-блок 2).

Значит, нормальный цикл работы лифта в рассматриваемом фрагменте реализуется либо за четыре такта, либо за шесть.

Таким образом, диаграмма достижимости может помочь качественно оценить функционирование проектируемой системы уже в стадии моделирования.

Другим важным свойством сети Петри является безопасность.

Позиция сети называется безопасной, если число фишек в ней никогда не превышает 1. Сеть Петри называется безопасной, если безопасны все позиции сети. Любая позиция сети может иметь только два значения: либо 0, либо 1.

Безопасность является важным свойством при моделировании аппаратного обеспечения системы управления, так как безопасная позиция может быть интерпретирована стандартным цифровым устройством – триггером. Это значит, что свойство безопасности можно применить ля оценки надежности проектируемой системы управления, особенно при ее аппаратной реализации.

Анализ сети Петри, моделирующей работу лифтовой системы (рисунок 5) на безопасность показывает, что все позиции сети являются безопасными. Начальное состояние сети с маркировкой позиции P_1 было обосновано и проанализировано выше, при описании графа сети Петри. Тот факт, что построенная сеть для моделирования работы лифта является формально безопасной, говорит об адекватности и надежности разработанной модели.

Следующим важным критерием работы сети является ее анализ на возможность параллелизма процессов.

Параллелизм (или одновременность) может быть представлен случаем, когда два независимых параллельных процесса описаны сетями Петри. В этом случае простая операция объединения двух сетей, приводит к составной сети Петри, в которой два процесса можно считать параллельными. В примере сети, представленной на рисунке 5, параллельные процессы отсутствуют, а есть лишь различные, возможные варианты разветвления процесса управления лифтовой системы.

Продолжая анализ сети для нашего примера, можно убедится в том, что сеть не имеет конфликтов, поскольку в сети не существует ситуации, когда два перехода приводят к противоречию в работе позиций, которые они обслуживают.

Отсутствие параллельных процессов в рассматриваемой сети исключает также и анализ способов синхронизации сети, поскольку синхронизация нужна в случае взаимодействия параллельных процессов в сети.

Кратко отметим, что синхронизация в сети Петри имеет специфический характер и связана не с привязкой моделируемого процесса с осью времени, а связана с процедурой распределения одного общего ресурса, на который

претендуют несколько процессов (в минимальном случае – два процесса). При этом каждый из процессов имеет одинаковые права на один ресурс. Вопрос о том – какой из претендующих процессов имеет более высокий приоритет на использование ресурса решается либо, исходя из особенностей реальных процессов, моделируемых сетью, либо определяется какой-то произвольной процедурой, имеющей случайный, вероятностный характер.

Эта процедура формально решается процессом присвоения фишек определенным позициям.

Таким образом, возможности теории сетей Петри позволяют произвести развернутый анализ проектируемых систем управления еще на стадии моделирования реальных процессов управления. Рассмотренные в этой главе дискретные модели для формализации процессов управления не являются единственными, а скорее наиболее распространенными.

Домашнее задание. Построить дерево достижимости для самостоятельной задачи по проектированию системы управления реального объекта, дать пояснения к дереву достижимости.

Работа 6

ОСНОВНЫЕ КОМПОНЕНТЫ ЯЗЫКА UML

UML представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем. Он является простым и мощным средством моделирования, который может быть эффективно использован для построения концептуальных, логических и графических моделей сложных систем самого различного целевого назначения. Этот язык вобрал в себя наилучшие качества и опыт методов программной инженерии, которые с успехом использовались на протяжении последних лет при моделировании больших и сложных систем.

Язык UML основан на некотором числе базовых понятий, которые могут быть изучены и применены большинством программистов и разработчиков, знакомых с методами объектно-ориентированного анализа и проектирования. При этом базовые понятия могут комбинироваться и расширяться таким образом, что специалисты объектного моделирования получают возможность самостоятельно разрабатывать модели больших и сложных систем в самых различных областях приложений.

Конструктивное использование языка UML основывается на понимании общих принципов моделирования сложных систем и особенностей процесса объектно-ориентированного анализа и проектирования в частности. Выбор выразительных средств для построения моделей сложных систем предопределяет те задачи, которые могут быть решены с использованием данных моделей. При этом одним из основных принципов построения моделей сложных систем является принцип абстрагирования, который предписывает включать в модель только те аспекты проектируемой системы, которые имеют непосредственное отношение к выполнению системой своих функций или своего целевого предназначения. При этом все второстепенные детали

опускаются, чтобы чрезмерно не усложнять процесс анализа и исследования полученной модели.

Другим принципом построения моделей сложных систем является принцип многомодельности. Этот принцип представляет собой утверждение о том, что никакая единственная модель не может с достаточной степенью адекватности описывать различные аспекты сложной системы. Феномен сложной системы как раз и состоит в том, что никакая ее единственная модель не является достаточной для адекватного выражения всех особенностей моделируемой системы.

Применительно к методологии ООАП это означает, что достаточно полная модель сложной системы представляет собой некоторое число взаимосвязанных представлений (views), каждое из которых адекватно отражает некоторый аспект поведения или структуры системы. При этом наиболее общими представлениями сложной системы принято считать статическое и динамическое представления, которые в свою очередь могут подразделяться на другие, более частные представления.

Еще одним принципом прикладного системного анализа является принцип иерархического построения моделей сложных систем. Этот принцип предписывает рассматривать процесс построения модели на разных уровнях абстрагирования или первоначальная модель сложной системы имеет наиболее общее представление (метапредставление). Такая модель строится на начальном этапе проектирования и может не содержать многих деталей и аспектов моделируемой системы.

Таким образом, процесс ООАП можно представить как поуровневый спуск от наиболее общих моделей и представлений концептуального уровня к более частным и детальным представлениям логического и физического уровня. При этом на каждом из этапов ООАП данные модели последовательно дополняются все большим количеством деталей, что позволяет им более адекватно отражать различные аспекты конкретной реализации сложной

системы. Общая схема взаимосвязей моделей ООАП представлена на рисунке 6.

Рисунок 6 – Общая схема взаимосвязей моделей и представлений сложной системы в процессе объектно-ориентированного анализа и проектирования

Название «физическая модель» в терминологии ООАП и языка UML отличается от общепринятой трактовки этого термина в общей классификации моделей систем. В последнем случае под физической моделью системы понимают некоторую материальную конструкцию, обладающую свойствами подобия с формой оригинала. Примерами таких моделей могут служить модели технических систем (самолетов, кораблей), архитектурных сооружений (зданий, микрорайонов). Что касается использования этого термина в ООАП и языке UML, то здесь физическая модель отражает компонентный состав проектируемой системы с точки зрения ее реализации на некоторой технической базе и вычислительных платформах конкретных производителей.

Язык UML предназначен для решения ряда задач.

1. Представить в распоряжение пользователей легко воспринимаемый и выразительный язык визуального моделирования, специально предназначенный для разработки и документирования моделей сложных систем самого различного целевого назначения.

Речь идет о том, что важным фактором дальнейшего развития и повсеместного использования методологии ООАП является интуитивная ясность и понятность основных конструкций соответствующего языка моделирования. Язык UML включает в себя не только абстрактные конструкции, для представления метамоделей систем, но и целый ряд конкретных понятий, имеющих вполне определенную семантику. Это позволяет языку UML одновременно достичь не только универсальности представления моделей для самых различных приложений, но и возможности описания достаточно тонких деталей реализации этих моделей применительно к конкретным системам.

Практика системного моделирования показала, что абстрактного описания языка на некотором метауровне недостаточно для разработчиков, которые ставят своей целью реализацию проекта системы в конкретные сроки. В настоящее время имеет место некоторый концептуальный разрыв между общей методологией моделирования сложных систем и конкретными инструментальными средствами быстрой разработки приложений. Именно этот разрыв и призван заполнить язык UML.

Отсюда вытекает важное следствие – для адекватного понимания базовых инструкций языка UML важно не только владеть некоторыми навыками объектно-ориентированного программирования, но и хорошо представлять себе общую проблематику процесса разработки моделей систем. Именно интеграция этих представлений образует новую парадигму ООАП, практическим следствием и центральным стержнем которой является язык UML.

2. Снабдить исходные понятия языка UML возможностью расширения и специализации для более точного представления моделей систем в конкретной предметной области.

Хотя язык UML является формальным языком спецификаций, формальность его описания отличается от синтаксиса как традиционных формально-логических языков, так и известных языков программирования. Это становится возможным по той причине, что в самом описании языка UML заложен механизм расширения базовых понятий, который является самостоятельным элементом и имеет собственное описание в форме правил расширения.

3. Описание языка UML должно поддерживать такую спецификацию моделей, которая не зависит от конкретных языков программирования и инструментальных средств проектирования программных систем.

Речь идет о том, что конструкции языка UML не должны зависеть от особенностей их реализации в известных языках программирования. Другими словами, хотя отдельные понятия языка UML семантически связаны с последними, их жесткая интерпретация в форме конструкций программирования не может быть признанной корректной.

С другой стороны, язык UML должен обладать потенциальной возможностью реализации своих конструкций на том или ином языке программирования. Конечно, в первую очередь имеются в виду языки, поддерживающие концепцию ООП, такие как C++, Java, Object Pascal. Именно это свойство языка UML делает его современным средством решения задач моделирования сложных систем. В то же время, предполагается, что для программной поддержки конструкций языка UML могут быть разработаны специальные инструментальные CASE – средства. Наличие последних имеет принципиальное значение для широкого распространения и использования языка UML.

4. Описание языка UML должно включает в себя семантический базис для понимания общих особенностей ООАП.

Говоря об этой особенности, имеют в виду самодостаточность языка UML для понимания не только его базовых конструкций, но что не менее важно – понимания общих принципов ООАП. В этой связи необходимо

отметить, что поскольку язык UML не является языком программирования, а служит средством для решения задач объектно-ориентированного моделирования систем, описание языка UML должно по возможности включать в себя все необходимые понятия для ООАП. Без этого свойства язык UML может оказаться бесполезным и невостребованным большинством пользователей, которые не знакомы с проблематикой ООАП сложных систем.

5. Способствовать распространению объектных технологий и соответствующих понятий ООАП.

Использование языка UML для решения всевозможных практических задач будет только способствовать его дальнейшему совершенствованию, а значит и дальнейшему развитию объектных технологий и практики ООАП.

6. Интегрировать в себя новейшие и наилучшие достижения практики ООАП.

Язык UML непрерывно совершенствуется разработчиками, и основой этой работы является его дальнейшая интеграция с современными технологиями моделирования. При этом различные методы системного моделирования получают свое прикладное осмысление в рамках ООАП. В последующем эти методы могут быть включены в состав языка UML в форме дополнительных базовых понятий, наиболее адекватно и полно отражающие наилучшие достижения практики ООАП.

Домашнее задание. Построить модель сложной системы для выбранного объекта управления, дать пояснения в терминах UML.

Работа 7
ОБЩАЯ СТРУКТУРА ЯЗЫКА UML

С самой общей точки зрения описание языка UML состоит из двух взаимодействующих частей:

семантика языка UML. Представляет собой некоторую метамодель, которая определяет абстрактный синтаксис и семантику понятий объектного моделирования на зыке UML;

нотация языка UML. Представляет собой графическую нотацию для визуального представления семантики языка UML.

Абстрактный синтаксис и семантика языка UML описываются с использованием некоторого подмножества нотации UML. В дополнении к этому, нотация UML описывает соответствие, или отображение графической нотации в базовые понятия семантики. Таким образом, с функциональной точки зрения эти две части дополняют друг друга. При этом семантика языка UML описывается на основе некоторой метамодели, имеющей три отдельных представления: абстрактный синтаксис, правила корректного построения выражений и семантику. Рассмотрение семантики языка UML предполагает некоторый «полуформальный» стиль изложения, который объединяет естественный и формальный языки для представления базовых понятий и правил их расширения.

Семантика определяется для двух видов объектных моделей: структурных моделей и моделей поведения. Структурные модели, известные также как статические модели, описывают структуру сущностей или компонентов некоторой системой, включая их классы, интерфейсы, атрибуты и отношения. Модели поведения, называемые иногда динамическими моделями, описывают поведение или функционирование объектов системы, включая их методы, взаимодействие и сотрудничество между ними, а также процесс изменения состояний отдельных компонентов и системы в целом.

Для решения столь широкого диапазона задач моделирования разработана достаточно полная семантика для всех компонентов графической нотации. Требования семантики языка UML конкретизируются при построении отдельных видов диаграмм. Нотация языка UML включает в себя описание отдельных семантических элементов, которые могут применяться при построении диаграмм.

Формальное описание самого языка UML основывается на некоторой общей иерархической структуре модельных представлений, состоящей из четырех уровней:

мета-метамодель;

метамодель;

модель;

объекты пользователя.

Уровень мета-метамодели образует исходную формально-логическую основу для всех метамодельных представлений. Главное предназначение этого уровня состоит в том, чтобы определить язык для спецификации метамодели. Мета-метамодель определяет модель языка UML на самом высоком уровне абстракции и является наиболее компактным ее описанием. С другой стороны, мета-метамодель может специфицировать несколько метамоделей, чем достигается потенциальная гибкость включения дополнительных понятий. Примерами понятий этого уровня служат метакласс, метаатрибут, метаоперация.

Следует отметить, что семантика мета-метамодели не входит в описание языка UML. С одной стороны, это делает язык UML более простым для изучения, поскольку не требует знания общей теории формальных языков и формальной логики. С другой стороны, наличие мета-метамодели придает языку UML черты формально-логического язык, которые необходимы ему для того, чтобы обладать свойством непротиворечивости. Если эти особенности могут представляться мало интересными для многих программистов, то разработчики инструментальных средств никак не могут их игнорировать.

Метамодель является экземпляром или конкретизацией мета-метамодели. Главная задача этого уровня – определить язык для спецификации моделей. Этот уровень является более конструктивным, чем предыдущий, поскольку обладает более развитой семантикой базовых понятий. Все основные понятия языка UML – это понятия уровня метамодели. Примеры таких понятий: класс, атрибут, операция, компонент, ассоциация и многие другие.

Модель в контексте языка UML является экземпляром метамодели в том смысле, что любая конкретная модель системы должна использовать только понятия метамодели, конкретизировав их применительно к данной ситуации. Это уровень для описания информации о конкретной предметной области. Однако если для построения модели используются понятия языка UML, то необходима полная согласованность понятий уровня модели с базовыми понятиями языка UML уровня метамодели. Примерами понятий уровня модели могут служить, например, имена полей проектируемой базы данных, такие как: имя и фамилия сотрудника, возраст, должность, адрес, телефон. При этом данные понятия используются лишь как имена соответствующих информационных атрибутов.

Конкретизация понятий модели происходит на уровне объектов пользователя. При этом совокупность объектов пользователя рассматривается как отдельный экземпляр модели, поскольку содержит конкретную информацию относительно того, чему в действительности соответствуют те или иные понятия модели. Примером объекта может служить следующая запись в проектируемой базе данных: "Илья Петров, 30 лет, иллюзионист, ул. Невидимая, 10-20, 100-0000".

Описание семантики языка UML предполагает рассмотрение базовых понятий только уровня метамодели, который представляет собой пример или частный случай уровня мета-метамодели. Метамодель UML является по своей сути, скорее логической моделью, чем физической или моделью реализации. Особенность, логической модели заключается в том, что она концентрирует внимание на декларативной или концептуальной семантике, опуская детали

конкретной физической реализации моделей. При этом отдельные реализации, использующие данную логическую метамодель, должны быть согласованы с ее семантикой, а также поддерживать возможности импорта и экспорта отдельных логических моделей.

В то же время, логическая метамодель может быть реализована различными способами для обеспечения требуемого уровня производительности и надежности соответствующих инструментальных средств. В этом заключается недостаток логической модели, которая не содержит, на уровне семантики требований, обязательных для ее эффективной последующей реализации. Однако согласованность метамодели с конкретными моделями реализации является обязательной для всех разработчиков программных средств, обеспечивающих поддержку языка UML.

Метамодель языка UML имеет довольно сложную структуру, которая включает в себя около 90 метаклассов, более 100 метаассоциаций и 50 стереотипов, число которых возрастает с появлением новых версий языка. Чтобы справится с этой сложностью языка UML, все его элементы организованы специальным образом в логически связанные подмножества, получившие названия пакетов. Поэтому рассмотрение языка UML на метамодельном уровне заключается в описании трех его наиболее общих логических блоков или пакетов: основные элементы, элементы поведения и управление моделями.

Для описания языка UML используются средства самого языка, и одним из таких средств является пакет. В общем случае пакет служит для группировки элементов модели. При этом сами элементы модели, которыми могут быть производственные сущности, отнесенные к одному пакету, выступают в роли единого целого. Пакеты, так же как и другие элементы модели, могут быть вложены в другие пакеты. Важной особенностью языка UML является тот факт, что все виды элементов модели UML могут быть организованы в пакеты.

Пакет - основной способ организации элементов модели в языке UML. Каждый пакет владеет всеми своими элементами, т. е. теми элементами, ко-

торые включены в него. Про соответствующие элементы пакета говорят, что они принадлежат пакету или входят в него. При этом каждый элемент может принадлежать только одному пакету. В свою очередь, одни пакеты могут быть вложены в другие пакеты. В этом случае первые называются подпакетами, поскольку все элементы подпакета будут принадлежать более общему пакету. Тем самым для элементов модели задается отношение вложенности пакетов, которое представляет собой иерархию.

Для графического представления иерархий могут использоваться графы специального вида, которые называются деревьями. Однако в языке UML эти графические обозначения настолько модифицированы, что соответствующие ассоциации с общетеоретическими понятиями могут представлять определенную трудность. Тем не менее, подчеркивается важность умения ассоциировать специальные конструкции языка UML с соответствующими понятиями теории множеств и системного моделирования, что, в некотором смысле, формирует стиль мышления системного аналитика. В противном случае не исключены досадные ошибки не только на начальном этапе концептуализации предметной области, но и в процессе построения различных представлений систем.

В языке UML для визуализации пакетов разработана специальная символика или графическая нотация, которой мы и будем пользоваться в дальнейшем. Именно с описания этой системы обозначений мы приступим к изучению основных элементов данного языка.

Для графического изображения пакетов на диаграммах применяется специальный графический символ - большой прямоугольник с небольшим прямоугольником, присоединенным к левой части верхней стороны первого (рисунок 7). Можно сказать, что визуально символ пакета напоминает пиктограмму папки в популярном графическом интерфейсе. Внутри большого прямоугольника может записываться информация, относящаяся к данному пакету. Если такой информации нет, то внутри большого прямоугольника записывается имя пакета, которое должно быть уникальным в пределах рас-

сматриваемой модели (рисунок 7, а). Если же такая информация имеется, то имя пакета записывается в верхнем маленьком прямоугольнике (рисунок 7, б).

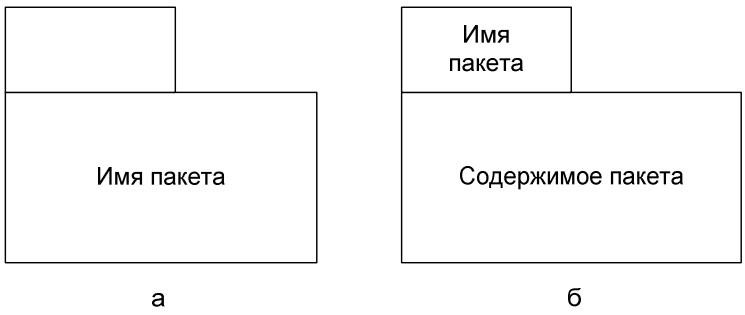

Рисунок 7 - Графическое изображение пакетов в языке UML

Говоря об имени пакета, следует остановиться на общем соглашении об именах в языке UML. В данном случае именем пакета может быть строка текста, содержащее любое число букв, цифр и некоторых специальных знаков. С целью удобства спецификации пакетов принято в качестве их имен использовать существительные, возможно, с пояснительными словами, например, контроллер, графический интерфейс, форма ввода данных.

Перед именем пакета помещается строка текста, содержащая некоторое ключевое слово. Такими ключевыми словами являются заранее определенные в языке UML слова, которые получили название стереотипов. Стереотипами для пакетов являются слова facade, framework, stub и topLevel. В качестве содержимого пакета могут выступать имена его отдельных элементов и их свойства, такие как видимость элементов за пределами пакета.

Конечно, сами по себе пакеты могут найти ограниченное применение, поскольку содержат лишь информацию о входящих в их состав элементах модели. Не менее важно представить графически отношения, которые могут иметь место между отдельными пакетами. Как и в теории графов, для визуализации отношений в языке UML применяются отрезки линий, внешний вид которых имеет смысловое содержание.

Одним из типов отношений между пакетами является отношение вложенности или включения пакетов друг в друга. С одной стороны, в языке

UML это отношение может быть изображено без использования линий простым размещением одного пакета-прямоугольника внутри другого пакета-прямоугольника (рисунок 8). Так, в данном случае пакет с именем Пакет_1 содержит в себе два подпакета: Пакет_2 и Пакет_3. На рисунке 9 показан пример вложенности пакетов с помощью визуализации отношения включения.

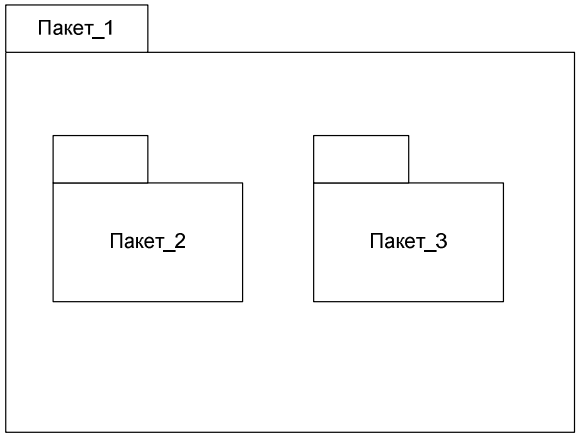

Рисунок 8 - Графическое изображение вложенности пакетов друг в друга

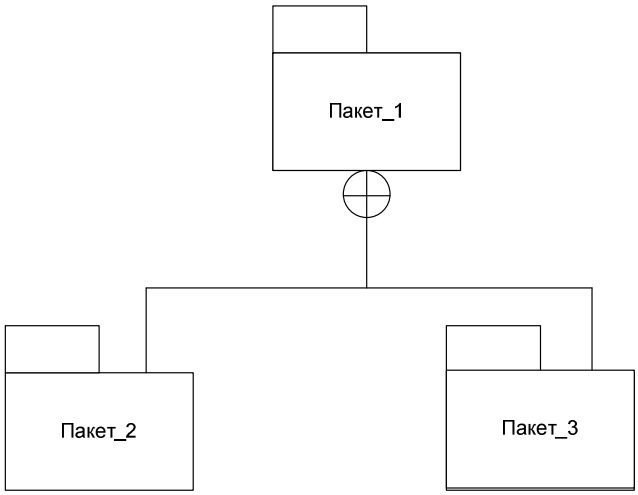

Рисунок 9 - Графическое изображение вложенности пакетов друг в друга с помощью явной визуализации отношения включения

Пакет Элементы ядра специфицирует базовые конструкции, требуемые для описания исходной метамодели, и определяет архитектурный "скелет" для присоединения дополнительных конструкций языка, таких как метаклассы, метаассоциации и метаатрибуты. Хотя пакет Элементы ядра содержит се-

мантику, достаточную для определения всей оставшейся части языка UML, он не является мета-метамоделью UML.

В этот пакет входят основные метаклассы языка UML: класс (Class), атрибут (Attribute), ассоциация (Association), ассоциация-класс (AssociationClass), конец ассоциации (AssociationEnd), свойство поведения (BehavioralFeature), классификатор (Classifier), ограничение (Constraint), тип данных (DataType), зависимость (Dependency), элемент (Element), право на элемент (ElementOwnership), свойство (Feature), обобщение (Generalization), элемент отношения обобщения (GeneralizableEIement), интерфейс (Interface), метод (Method), элемент модели (ModelElement), пространство имен (Namespace), операция (Operation), параметр (Parameter), структурное свойство (StructuralFeature), правила правильного построения выражений (Well-formedness rules).

Пакет Механизмы расширения специфицирует порядок включения в модель элементов с уточненной семантикой, а также модификацию отдельных компонентов языка UML для более точного отражения специфики моделируемых систем. Механизм расширения определяет семантику для стереотипов, ограничений и помеченных значений. Хотя, язык UML обладает богатым множеством понятий и нотаций для моделирования типичных программных систем, реально разработчик может столкнуться с необходимостью включить в модель дополнительные свойства или нотации, которые не определены явно в языке UML. При этом разработчики часто сталкиваются с необходимостью включения в модель графической информации, такой, например, как дополнительные значки и украшения.

Для этой цели в языке UML предусмотрены три механизма расширения, которые могут использоваться совместно или раздельно для определения новых элементов модели отличающимися семантикой, нотацией и свойствами, от специфицированных в метамодели языка UML элементов. Такими механизмами являются: ограничение (Constraint), стереотип (Stereotype) и помеченное значение (TaggedValue).

Таким образом, механизмы расширения языка UML предназначены для выполнения следующих задач:

уточнения или специализации достаточно общих модельных элементов при разработке конкретных моделей на языке UML;

определения таких расширений языка UML, которые зависят от специфики моделируемого процесса или от языка реализации программного кода;

присоединения произвольной семантической или несемантической информации к элементам модели.

Наиболее важные из встроенных механизмов расширения основываются на понятии стереотип. Стереотипы обеспечивают некоторый способ классификации модельных элементов на уровне объектной модели и возможность добавления в язык UML "виртуальных" метаклассов с новыми атрибутами и семантикой. Другие встроенные механизмы расширения основываются на понятии списка свойств, содержащего помеченные значения и ограничения. Эти механизмы обеспечивают пользователю возможность включения дополнительных свойств и семантики непосредственно в отдельные элементы модели.

Пакет Типы данных специфицирует различные типы данных, которые могут использоваться в языке UML. Этот пакет имеет более простую по сравнению с другими пакетами внутреннюю структуру и описание, поскольку предполагается, что семантика соответствующих понятий хорошо известна.

В метамодели UML типы данных используются для объявления типов атрибутов классов. Они записываются в форме строк текста на диаграммах и не имеют отдельного значка "тип данных". Благодаря этому происходит уменьшение размеров диаграмм без потери информации. Однако каждая из одинаковых записей для некоторого типа данных должна соответствовать одному и тому же типу данных в модели. При этом типы данных, используемые в описании языка UML, могут отличаться от типов данных, которые определяет разработчик для своей модели на языке UML. Типы данных в последнем случае

будут являться частным случаем или экземплярами метакласса типы данных, который определен в метамодели.

При задании типа данных наиболее часто применяется неформальная конструкция, которая получила название перечисления. Пакет Элементы поведения является самостоятельной компонентой языка UML и, как следует из его названия, специфицирует динамику поведения в нотации UML. Пакет Элементы поведения состоит из подпакетов: Общее поведение, Кооперации Варианты использования, Конечные автоматы, Графы деятельности и Действия (рисунок 10). Ниже дается краткая характеристика каждого из этих подпакетов.

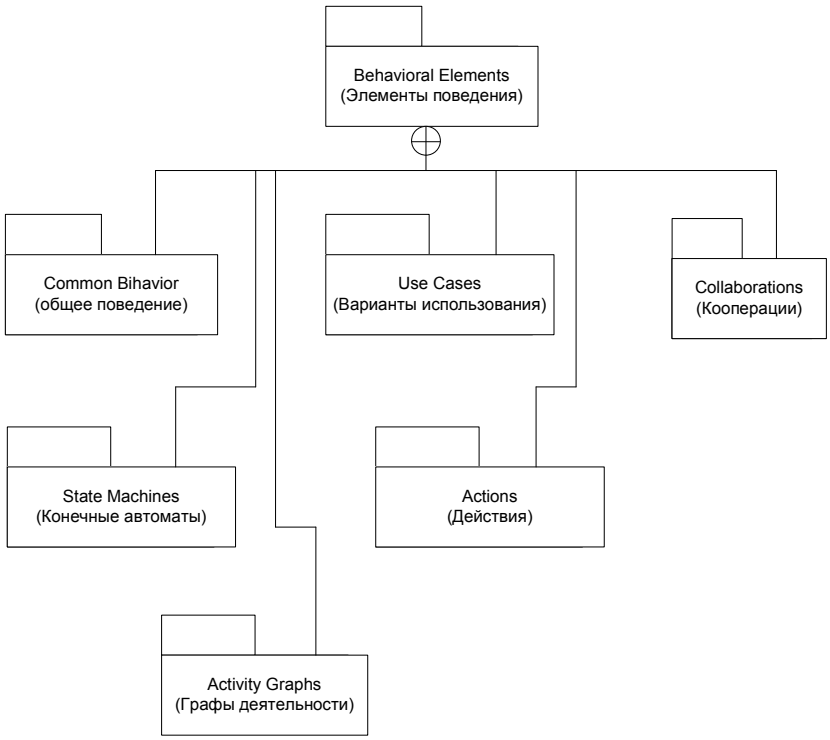

Рисунок 10 – Подпакеты пакета Элементы поведения языка UML

Пакет Общее поведение является наиболее фундаментальным из всех подпакетов и определяет базовые понятия ядра, необходимые для всех элементов поведения. В этом пакете специфицирована семантика для динамических элементов, которые включены в другие, подпакеты элементов поведения. В пакет Общее поведение входит достаточно большое число элементов, таких как: экземпляр (Instance), объект (Object), исключение

(Exception), связь (Link), сигнал (Signal), значение данных (DataValue), процедура (Procedure), связь атрибутов (AttributeLink) и т.д.

Наиболее важным понятием пакета Общее поведение является объект. Под объектом в языке UML понимается отдельный экземпляр или пример класса, структура и поведение которого полностью определяется порождающим этот объект классом. Предполагается, что все без исключения объекты, порожденные одним и тем же классом, имеют совершенно одинаковую структуру и поведение, хотя каждый из этих объектов может иметь свое собственное множество связей атрибутов. При этом каждая связь атрибута относится к некоторому экземпляру, обычно к значению данных. Это множество может быть модифицировано согласно спецификации отдельного атрибута в описании класса.

Рассматривая данный пакет, следует отметить, что в языке UML под поведением понимается не только процесс изменения атрибутов объектов в результате выполнения операций над их значениями, но и такие процедуры, как создание и уничтожение самих объектов. При этом динамика взаимодействия объектов, которая определяет их поведение, описывается с помощью специальных понятий, таких как сигналы и действия.

Пакет Кооперации специфицирует контекст поведения при использовании элементов модели для выполнения отдельной задачи. В нем задается семантика понятий, которые необходимы для ответа на вопрос "Как различные элементы модели взаимодействуют между собой с точки зрения структуры?". Этот пакет использует конструкции, определенные в пакетах Основные элементы языка UML и Общее поведение.

В частности, в пакет Кооперации входят элементы: кооперация (Collaboration), взаимодействие (Interaction), сообщение (Message), роль ассоциации (AssociationRole), роль классификатора (ClassifierRole), роль конца ассоциации (AssociationEndRole). Как можно догадаться из названия пакета, его элементы непосредственно используются при построении диаграмм кооперации. Понятие кооперации имеет важное значение для представления

взаимодействия элементов модели с точки зрения классификаторов и ассоциаций.

Пакет Варианты использования специфицирует поведение при включении в модель специальных конструкций, которые в языке UML называются актерами и вариантами использования. Эти понятия служат для определения функциональности моделируемой сущности, такой как система. Особенность элементов этого пакета состоит в том, что они используются для первоначального определения поведения сущности без спецификации ее внутренней структуры.

Объединение в языке UML средств концептуализации исходных требований к проектируемой системе и структуризации ее внутренних компонентов с достаточно богатой семантикой применяемых для этого элементов имеет важное значение для построения адекватных моделей сложных систем. Действительно, ограниченность традиционных моделей состоит в том, что они не позволяют одновременно описывать статические или структурные свойства системы и динамику ее поведения. Попытки совместного решения данных проблем сталкиваются с отсутствием единой символики для обозначения близких по смыслу системных понятий. Язык UML удачно выделяет базовые понятия, которые необходимы при построении таких моделей. Более того, если этих понятий окажется недостаточно для разработки какого-то конкретного проекта, то сам разработчик может расширить базовые понятия и даже включить в модель собственные конструкции, согласованные с метамоделью языка UML.

В пакете Варианты использования кроме уже упомянутых элементов актер (Actor) и вариант использования (UseCase) входят: расширение (Extend), точка расширения (ExtensionPoint), включение (Include) и экземпляр варианта использования (UseCaseInstance).

Пакет Конечные автоматы специфицирует поведение при построении моделей с использованием систем переходов для конечного множества состояний. В нем определено множество понятий, которые необходимы для

представления поведения модели в виде, дискретного пространства с конечным числом состояний и переходов.

Формализм конечного автомата, который используется в языке UML, отличается от формализма теории автоматов своей объектной ориентацией. Конечные автоматы являются основным средством моделирования поведения различных элементов языка UML. Они, например, могут использоваться для моделирования поведения индивидуальных сущностей, таких как экземпляры классов или объектов, а также для спецификации взаимодействий между сущностями, такими как кооперации. Формализм конечных автоматов дополнительно обеспечивает семантический базис для графов деятельности, которые являются частным случаем конечного автомата.

В пакет Конечные автоматы входят элементы: состояние (State), переход (Transition), событие (Event), конечный автомат (StateMachine); простое состояние (SimpleState), составное состояние (CompositeState), псевдосостояние (PseudoState), конечное состояние (FinalState), сторожевое условие (Guard) и некоторые другие.

Одним из ключевых понятий: при моделировании динамических свойств систем является состояние. При этом под состоянием в языке UML понимается абстрактный метакласс, используемый для моделирования ситуации или процесса, в ходе которых имеет место (обычно неявное) выполнение некоторого инвариантного условия. Примером такого условия может быть состояние ожидания объектом выполнения некоторого внешнего события, например, запроса или передачи управления. С другой стороны, состояние может использоваться для моделирования динамических условий, таких как процесс выполнения некоторой деятельности. В этом случае момент начала выполнения деятельности является переходом объекта в соответствующее состояние.

В этом пакете специфицированы понятия, которые могут быть использованы для построения моделей процессов с использованием нетриггерных переходов. В нем определено множество понятий, которые

необходимы для представления логики протекания процессов и выполнения процедур или алгоритмов, включая бизнес-процессы и документооборот компаний и фирм.

В пакет Графы деятельности входят элементы: граф деятельности (ActivityGraph), разбиение (Partition), состояние действия (ActionState), состояние поддеятельности (SubactivityState), состояние вызова (CallState) и некоторые другие.

Пакет Действия является новым пакетом языка UML и специфицирует синтаксис и семантику выполняемых действий и процедур, включая семантику времени их выполнения. Этот пакет в свою очередь состоит из нескольких подпакетов, в которых определяются те или иные конкретные действия. Эти действия связаны с выполнением различных процедур, таких как создание и уничтожение объектов (CreateObjectAction и DestroyObjectAction), создание и уничтожение связей (CreateLinkAction и DestroyLinkAction), вычисление (Computation Actions), чтение и запись (Read Write Actions), передача сообщений (Messaging Actions) и т. д.

Пакет Управление моделями (Model Management) специфицирует базовые элементы языка UML, которые необходимы для формирования всех модельных представлений. Именно в нем определяется семантика модели (Model), пакета (Package) и подсистемы (Subsystem). Эти элементы служат своеобразными контейнерами для группировки других элементов модели.

Пакет является метаклассом в языке UML и предназначен для организации других элементов модели, таких как другие пакеты, классификаторы и ассоциации. Пакет может также содержать ограничения и зависимости между элементами модели в самом пакете. Предполагается, что каждый элемент пакета имеет видимость только внутри данного пакета. Это означает, что за пределами пакета никакой его элемент не может быть использован, если нет дополнительных указаний на импорт или доступ к отдельным элементам пакета. Со своей стороны, пакеты со всем своим содержимым определены в некотором пространстве имен, которое определяет

единственность использования имен всех элементов модели. Другими словами, имя каждого элемента модели должно быть единственным в некотором пространстве имен, которое, являясь само элементом модели, может быть вложено в более общее пространство имен.

Модель является подклассом пакета и представляет собой абстракцию физической системы, которая предназначена для вполне определенной цели. Именно эта цель предопределяет те компоненты, которые должны быть включены в модель, и те, рассмотрение которых не является обязательным. Другими словами, модель отражает релевантные аспекты физической системы, оказывающие непосредственное влияние на достижение поставленной цели. В прикладных задачах цель обычно задается в форме исходных требований к системе, которые, в свою очередь, в языке UML записываются в виде вариантов использования системы.

В языке UML для одной и той же физической системы могут быть определены различные модели, каждая из которых специфицирует систему различных точек зрения. Примерами таких моделей являются логическая модель, модель проектирования, модель вариантов использования и другие. При этом каждая такая модель имеет свою собственную точку зрения на физическую систему и свой собственный уровень абстракции. Модели, как и пакеты, могут быть вложенными друг в друга. Со своей стороны, пакет может включать в себя несколько различных моделей одной и той же системы, и в этом состоит один из важнейших механизмов разработки моделей на языке UML. В общем случае модель системы в контексте языка UML включает в себя модель анализа и модель проектирования, что с использованием обозначений пакетов может быть изображено следующим образом (рисунок 11).

Подсистема есть просто группировка элементов модели, которые специфицируют некоторое, простей шее поведение физической системы. В метамодели UML подсистема является подклассом, как пакета, так и классификатора. Элементы подсистемы делятся на две части — спецификацию поведения и его реализацию.

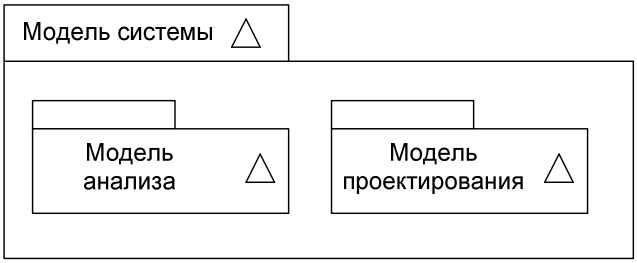

Рисунок 11 – Изображение модели системы в виде пакетов моделей анализа и проектирования

Для графического представления подсистемы применяется специальное обозначение — прямоугольник, как в случае пакета, но дополнительно разделенный на три секции (рисунок 12). При этом в верхнем маленьком прямоугольнике изображается символ, по своей форме напоминающий "вилку" и указывающий на подсистему. Имя подсистемы вместе с необязательными ключевым словом или стереотипом записывается внутри большого прямоугольника. Однако при наличии строк текста внутри большого прямоугольника имя подсистемы может быть записано рядом с обозначением "вилки".

Рисунок 12 – Графическое изображение подсистемы в языке UML

Домашнее задание. Построить графическое изображение подсистемы в языке UML для выбранного объекта управления.

Работа 8

СПЕЦИФИКА ОПИСАНИЯ МЕТАМОДЕЛИ ЯЗЫКА UML

Метамодель языка UML описывается на некотором полуформальном языке с использованием трех видов представлений:

абстрактный синтаксис;

правила правильного построения выражений;

семантика.

Абстрактный синтаксис представляет собой модель для описания некоторой части языка UML, предназначенной для построения диаграмм классов на основе описаний систем на естественном языке. Возможности абстрактного синтаксиса в языке UML довольно ограничены и имеют отношение только к интерпретации обозначений отдельных компонентов диаграмм, связей между компонентами и допустимых дополнительных обозначений. К элементам абстрактного синтаксиса относятся некоторые ключевые слова и значения отдельных атрибутов базовых понятий уровня метамодели, которые имеют фиксированное обозначение в виде текста на естественном языке.

Правила правильного построения выражений используются для задания дополнительных ограничений или свойств, которыми должны обладать те или иные компоненты модели. Поскольку исходным понятием 00ГГ является* понятие класса, его общими свойствами должны обладать все экземпляры, которые в этом смысле должны быть инвариантны друг другу. Для задания этих инвариантных свойств классов и отношений необходимо использовать специальные выражения некоторого формального языка, в рамках UML получившего название языка объектных ограничений (Object Constraint Language, OCL).

Семантика языка UML описывается в основном на естественном языке, но может включать в себя некоторые дополнительные обозначения, вытекающие из связей определяемых понятии с другими понятиями. Семантика по-

нятий раскрывает их смысл или содержание. Сложность описания семантики языка UML заключается именно в метамодельном уровне представлений его основных конструкций. С одной стороны, понятия языка UML имеют абстрактный характер (ассоциация, композиция, агрегация, сотрудничество, состояние). С другой стороны, каждое из этих понятий допускает свою конкретизацию на уровне модели (сотрудник, отдел, должность, стаж).

Сложность описания семантики языка UML вытекает из этой двойственности понятий. Здесь мы должны придерживаться традиционных правил изложения, поскольку понимание семантики носит индуктивный характер и требует для своей интерпретации примеров уровня модели и объекта. Иллюстрация абстрактных понятий на примере конкретных свойств и отношений, а также их значений позволяет акцентировать внимание на общих инвариантах этих понятий, что совершенно необходимо для понимания их семантики.

Таким образом, метамодель языка UML может рассматриваться как комбинация графической нотации (специальных обозначений), некоторого формального языка и естественного языка. При этом следует иметь в виду, что существует некоторый теоретический предел, который ограничивает описание метамодели средствами самой метамодели. Именно в подобных случаях используется естественный язык, обладающий наиболее выразительными возможностями.

Хотя сам термин "естественный язык" далеко не однозначен и порождает целый ряд дополнительных вопросов, здесь мы ограничимся его трактовкой в форме обычного текста на русском и, возможно, английском языках. Как бы ни хотелось некоторым из отечественных разработчиков, полностью избежать использования английского при описании языка UML не удастся. Тем не менее, если исключить написание стандартных элементов и некоторых ключевых слов, то во всех остальных случаях под естественным языком можно понимать русский без специальных оговорок.

Для придания формального характера моделям UML использование естественного языка должно строго соответствовать определенным правилам. Например, описание семантики языка UML может включать в себя фразы типа "Сущность А обладает способностью" или "Сущность Б есть сущность В". В каждом из этих случаев мы будем понимать смысл фраз, руководствуясь традиционным пониманием предложений русского языка. Однако этого может оказаться недостаточно для более формального представления знаний о рассматриваемых сущностях. Тогда необходимо дополнительно специфицировать семантику этих простых фраз, для чего рекомендуется использовать следующие правила:

явно указывать в тексте экземпляр некоторого метакласса. Речь идет о том, что в естественной речи мы часто опускаем слово "пример" или "экземпляр", говоря просто "класс". Так, фразу "Атрибут возраст класса сотрудник имеет значение 30 лет" следует записать более точно, а именно: "Атрибут возраст экземпляра класса сотрудник имеет значение 30 лет";

в каждый момент используется только то значение слова, которое приписано имени соответствующей конструкции языка UML. Все дополнительные особенности семантики должны быть указаны явным образом, без каких бы то ни было неявных предположений;

термины языка UML могут включать, только один из допустимых префиксов, таких как под-, супер- или мета-. При этом сам термин с префиксом записывается одним словом.

В дополнение к этому будут использоваться следующие правила выделения текста:

если используются ссылки на конструкции языка UML, а не на их представления в метамодели, следует применять обычный текст, без какого бы то ни было выделения;

имена метаклассов являются элементом нотации языка UML и представляют собой существительное и, возможно, присоединенное к нему прилагательное. В этом случае имя метакласса на английском записывается одним

словом с выделением каждой составной части имени заглавной буквой (например, ModelElement, StructuralFeature);

имена метаассоциаций и ассоциаций классов записываются аналогичным образом (например, ElementReference);

имена других элементов языка UML также записываются одним словом, но должны начинаться со строчной буквы (например, owned Element, all-Contents);

имена метаатрибутов, которые принимают булевы значения, всегда начинаются с префикса "is" (например, isAbstract);

перечислимые типы должны всегда заканчиваться словом "Kind" (например, AggregationKind);

при ссылках в тексте на метаклассы, метаассоциаций, метаатрибуты и т. д. должны всегда использоваться в точности те их имена, которые указаны в модели;

имена стандартных обозначений (стереотипов) заключаются в угловые кавычки и начинаются со строчной буквы (например, «type»).

Рассмотренные выше правила выделения текста имеют непосредственное отношение к англоязычным терминам языка UML.

при описании семантики языка UML все имена его стандартных элементов (метаклассов, метаассоциаций, метаатрибутов) допускается записывать на русском с дополнительным указанием оригинального имени на английском языке. При этом, хотя имена стандартных элементов могут состоять из нескольких слов, согласно сложившейся отечественной традиции, будем их записывать раздельно (например, класс ассоциации, элемент модели, пространство имен);

при разработке конкретных моделей систем в форме диаграмм языка UML целесообразно применять оригинальные англоязычные термины, придерживаясь описанных ранее правил (кроме, возможно, пояснительного текста на русском). Причина этой рекомендации вполне очевидна - последующая инструментальная реализация модели может оказаться не-

возможной, если не следовать оригинальным правилам выделения текста в языке UML. Это правило не распространяется на отдельные примеры и фрагменты диаграмм, которые приводятся в тексте книги с чисто иллюстративными целями и лишь раскрывают особенности использования стандартных элементов языка UML.

В рамках языка UML все представления о модели сложной системы фиксируются в виде специальных графических конструкций, получивших название диаграмм. В терминах языка UML определены следующие виды диаграмм:

диаграмма вариантов использования (use case diagram);

диаграмма классов (class diagram);

диаграммы поведения (behavior diagrams);

диаграммы взаимодействия (interaction diagrams);

диаграмма кооперации (collaboration diagram);

диаграмма последовательности (sequence diagram);

диаграмма состояний (statechart diagram);

диаграмма деятельности (activity diagram);

диаграммы реализации (implementation diagrams);

диаграмма компонентов (component diagram);

диаграмма развертывания (deployment diagram).

Из перечисленных диаграмм некоторые служат для обозначения двух и более других подвидов диаграмм.

Перечень этих диаграмм и их названия являются каноническими в том смысле, что представляют собой неотъемлемую часть графической нотации языка UML. Более того, процесс ООАП неразрывно связан с процессом построения этих диаграмм. При этом совокупность построенных таким образом диаграмм является самодостаточной в том смысле, что в них содержится вся информация, которая необходима для реализации проекта сложной системы.

Каждая из этих диаграмм детализирует и конкретизирует различные представления о модели сложной системы в терминах языка UML. При этом

диаграмма вариантов использования представляет собой наиболее общую концептуальную модель сложной системы, которая является исходной для построения всех остальных диаграмм. Диаграмма классов является, по своей сути, логической моделью, отражающей статические аспекты структурного построения сложной системы.

Диаграммы поведения также являются разновидностями логической модели, которые отражают динамические аспекты функционирования сложной системы. И, наконец, диаграммы реализации служат для представления физических компонентов сложной системы и поэтому относятся к ее физической модели. Таким образом, интегрированная модель сложной системы в нотации UML может быть представлена в виде совокупности указанных выше диаграмм (рисунок 13).

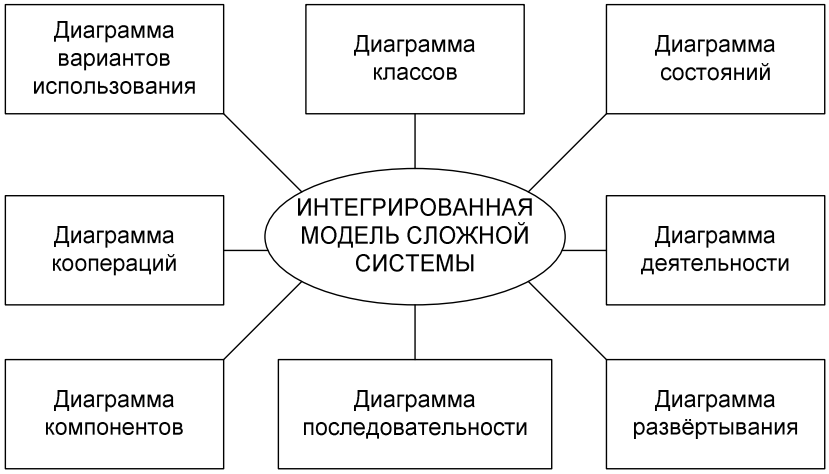

Рисунок 13 – Интегрированная модель сложной системы в нотации UML

Большинство перечисленных ранее диаграмм являются в своей основе графами специального вида, состоящими из вершин в форме геометрических фигур, которые связаны между собой ребрами или дугами. Поскольку информация, которую содержит в себе граф, имеет в основном топологический характер, ни геометрические размеры, ни расположение элементов диаграмм (за некоторыми исключениями, такими как диаграмма последовательностей, с метрической осью времени) не имеют принципиального значения.

Для диаграмм языка UML существуют три типа визуальных графических обозначений, которые важны с точки зрения заключенной в них информации.

- геометрические фигуры на плоскости, играющие роль вершин графов соответствующих диаграмм. При этом сами геометрические фигуры выступают в роли графических примитивов языка UML, а форма этих фигур (прямоугольник, эллипс) должна строго соответствовать изображению отдельных элементов языка UML (класс, вариант использования, состояние, деятельность). Графические примитивы языка UML имеют фиксированную семантику, переопределять которую пользователям не допускается. Графические примитивы должны иметь собственные имена, а возможно, и другой текст, который содержится внутри границ соответствующих геометрических фигур или, как исключение, вблизи этих фигур;

графические взаимосвязи, которые представляются различными линиями на плоскости. Взаимосвязи в языке UML обобщают понятие дуг и ребер из теории графов, но имеют менее формальный характер и более развитую семантику;

специальные графические символы, изображаемые вблизи от тех или иных визуальных элементов диаграмм и имеющие характер дополнительной спецификации (украшений).

Все диаграммы в языке UML изображаются с использованием фигур на плоскости. Однако некоторые из фигур (например, кубы) могут представлять собой двумерные проекции трехмерных геометрических тел, но и в этом случае они рисуются как фигура на плоскости.

Таким образом, в языке UML используется четыре основных вида конструкций.

графические фигуры на плоскости. Такие двумерные символы изображаются с помощью некоторых геометрических фигур и могут иметь различную высоту и ширину с целью размещения внутри этих фигур других конструкций языка UML. Наиболее часто внутри таких символов помещаются строки текста, которые уточняют семантику или фиксируют отдельные свойства

соответствующих элементов языка UML. Информация, содержащаяся внутри фигур, имеет важное, значение для конкретной модели проектируемой системы, поскольку регламентирует реализацию соответствующих элементов в программном коде;

пути, которые представляют собой последовательности из отрезков линий, соединяющих отдельные графические символы. При этом концевые точки отрезков линий должны обязательно соприкасаться с геометрическими фигурами, служащими для обозначения вершин диаграмм, как принято в теории графов. С концептуальной точки зрения путям в языке UML придается особое значение, поскольку они являются простыми топологическими сущностями. С другой стороны, отдельные части пути или сегменты могут не существовать сами по себе вне содержащего их пути. Пути всегда соприкасаются с другими графическими символами на обеих границах соответствующих отрезков линий. Другими словами, пути не могут обрываться на диаграмме линией, которая не соприкасается ни с одним графическим символом. Как уже отмечалось, пути могут иметь в качестве окончания или терминатора специальную графическую фигуру - значок, который изображается на одном из концов линий;

значки или пиктограммы. Значок представляет собой графическую фигуру фиксированного размера и формы. Она не может увеличивать свои размены, чтобы разместить внутри себя дополнительные символы. Значки могут размещаться как внутри других графических конструкций, так и вне них. Примерами значков могут служить окончания связей элементов диаграмм или некоторые другие дополнительные обозначения (украшения);

строки текста. Служат для представления различных видов информации в некоторой грамматической форме. Предполагается, что каждое использование строки текста должно соответствовать синтаксису в нотации языка UML, посредством которого может быть реализован грамматический разбор этой строки. Последний необходим для получения полной информации о модели. Например, строки текста в различных секциях обозначения класса могут

соответствовать атрибутам этого класса или его операциям. На использование строк накладывается важное условие - семантика всех допустимых символов должна быть заранее определена в языке UML или служить предметом его расширения в конкретной модели;

При графическом изображении диаграмм следует придерживаться основных рекомендаций.

каждая диаграмма должна служить законченным представлением соответствующего фрагмента моделируемой предметной области. Речь идет о том, что в процессе разработки диаграммы необходимо учесть все сущности, важные с точки зрения контекста данной модели и диаграммы. Отсутствие тех или иных элементов на диаграмме служит признаком неполноты модели и может потребовать ее последующей доработки;

все сущности на диаграмме модели должны быть одного концептуального уровня. Здесь имеется в виду согласованность не только имен одинаковых элементов, но и возможность вложения отдельных диаграмм друг в друга для достижения полноты представлений. В случае достаточно сложных моделей систем желательно придерживаться стратегии последовательного уточнения или детализации отдельных диаграмм;

вся информация о сущностях должна быть, явно представлена на диаграммах. Речь идет о том, что хотя в языке UML при отсутствии некоторых символов на диаграмме могут быть использованы их значения по умолчанию (например, в случае неявного указания видимости атрибутов и операций классов), необходимо стремиться к явному указанию свойств всех элементов диаграмм;

диаграммы не должны содержать противоречивой информации. Противоречивость модели может служить причиной серьезнейших проблем при ее реализации и последующем использовании на практике. Например, наличие замкнутых путей при изображении отношений агрегирования или композиции приводит к ошибкам в программном коде, который будет реализовывать соответствующие классы. Наличие элементов с одинаковыми

именами и различными атрибутами свойств, в одном пространстве имен, также приводит к неоднозначной интерпретации, и может служить источником проблем.

Наличие в инструментальных CASE-средствах встроенной поддержки визуализации различных диаграмм языка UML позволяет в некоторой степени исключить ошибочное использование тех или иных графических символов, а также контролировать уникальность имен элементов диаграмм. Однако, поскольку вся ответственность за окончательный контроль непротиворечивости-модели лежит на разработчике, необходимо помнить, что неформальный характер языка UML может служить источником потенциальных ошибок, которые в полном объеме вряд ли будут выявлены инструментальными средствами. Именно это обстоятельство требует от всех разработчиков глубокого знания нотации и семантики всех элементов языка UML.

диаграммы не следует перегружать текстовой информацией. Принято считать, что визуализация модели является наиболее эффективной, если она содержит минимум пояснительного текста. Как правило, наличие больших фрагментов развернутого текста служит признаком недостаточной проработанности модели или её неоднородности, когда в рамках одной модели представляется различная по характеру информация. Поскольку общая декомпозиция модели на отдельные типы диаграмм способна удовлетворить самые детальные представления разработчиков о системе, важно уметь правильно отображать те или иные сущности и аспекты моделирования в соответствующие элементы канонических диаграмм;

каждая диаграмма должна быть самодостаточной для правильной интерпретации всех ее элементов и понимания семантики всех используемых графических символов. Любые пояснительные тексты, которые не являются собственными элементами диаграммы (например, комментариями), не должны приниматься во внимание разработчиками. В то же время отдельные общие фрагменты могут уточняться или детализироваться на

других диаграммах этого же типа, образуя вложенные или подчиненные диаграммы. Таким образом, модель системы на языке UML представляет собой пакет иерархически вложенных диаграмм, детализация которых должна быть достаточной для последующей генерации программного кода, реализующего проект соответствующей системы;

количество типов диаграмм для конкретной модели приложения не является строго фиксированным. Речь идёт о том, что для простых приложений нет необходимости строить все без исключения типы диаграмм. Некоторые из них могут просто отсутствовать в проекте системы, и этот факт не будет считаться ошибкой разработчика. Например, модель системы может не содержать диаграмму развертывания для приложения выполняемого локально на компьютере пользователя. Важно понимать, что перечень диаграмм зависит от специфики конкретного проекта системы;

любая из моделей системы должна содержать только те элементы, которые определены в нотации языка UML. Имеется в виду требование начинать разработку проекта, используя только те конструкции, которые уже определены в метамодели UML. Как показывает практика, этих конструкций вполне достаточно для представления большинства типовых проектов программных систем. И только в случае отсутствия необходимых базовых элементов языка UML следует использовать механизмы их расширения для адекватного представления конкретной модели системы. При этом не допускается переопределение семантики тех элементов, которые отнесены к базовой нотации метамодели языка UML.

Как не вспомнить в этой связи известный афоризм, получивший название "бритва Оккама". Суть изречения средневекового ученого-схоласта в достаточно вольном переводе сводится к следующему: "Не плоди рассуждений больше сущности". Другими словами, нужно стремиться дополнительно не усложнять и без того сложные модели систем, а по возможности упрощать их за счет унификации обозначений и семантики базовых элементов.

Процесс построения отдельных типов диаграмм имеет свои особенности, которые тесно связаны с семантикой элементов этих диаграмм. Сам процесс ООАП в контексте языка UML получил специальное название - рациональный унифицированный процесс (Rational Unified Process, RUP). Концепция RUP и основные его элементы разработаны А. Джекобсоном в ходе его работы над языком UML.

При дословном переводе термина RUP теряется некоторая дополнительная семантическая окраска, связанная с двусмысленным толкованием английского Rational. Речь идет о другом варианте перевода - унифицированный процесс от фирмы Rational Software, сотрудниками которой являются с некоторых пор его разработчики, включая упомянутого выше А. Джекобсона.

Суть концепции RUP заключается в последовательной декомпозиции или разбиении процесса ООАП на отдельные этапы, на каждом из которых осуществляется разработка соответствующих типов канонических диаграмм модели системы. При этом изначальных этапах RUP, строятся логические представления статической модели структуры системы, затем - логические представления модели поведения, и лишь после этого - физические представления модели системы. Как нетрудно заметить, в результате RUP должны быть построены канонические диаграммы на языке UML, при этом последовательность их разработки может соответствовать их последовательной нумерации.

Домашнее задание. Построить и описать метамодель языка UML для выбранного объекта управления.

Работа 9

КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ЯЗЫКА UML

Реляционная модель данных в подавляющем большинстве случаев вполне достаточна для моделирования любых данных. Однако проектирование базы данных в терминах схемы отношений на практике может вызвать большие затруднения, т.к. в этой модели изначально не предусмотрены механизмы описания семантики предметной области. С этим связано появление семантических моделей данных, которые позволяют описать конкретную предметную область гораздо ближе к интуитивному пониманию и, в то же время, достаточно формальным образом.

Часто семантическое моделирование используется только на первой стадии проектирования базы данных. Концептуальная схема будущей БД строится на основе некоторой семантической модели, а затем вручную преобразуется к реляционной схеме.

Существуют методики, четко описывающие все этапы такого преобразования.

При таком подходе отсутствует потребность в дополнительных программных средствах, поддерживающих семантическое моделирование. Требуется только владение основами выбранной семантической модели и правилами преобразования концептуальной схемы в реляционную.

Следует заметить, что многие начинающие проектировщики баз данных недооценивают важность семантического моделирования вручную. Зачастую это воспринимается как дополнительная и излишняя работа. Эта точка зрения является абсолютно неверной. Во-первых, построение мощной и наглядной концептуальной схемы БД позволяет более полно оценить специфику моделируемой предметной области и избежать возможных ошибок на стадии проектирования схемы реляционной БД. Во-вторых, на этапе семантического моделирования производится очень важная документация (хотя бы в виде

вручную нарисованных диаграмм и комментариев к ним), которая может оказаться полезной не только при проектировании схемы реляционной БД, но и при эксплуатации, сопровождении и развитии уже заполненной БД.

Неоднократно приходилось наблюдать жизненные ситуации, когда отсутствие такого рода документации существенно затрудняло выполнение даже небольших изменений в схеме существующей реляционной БД. Конечно, это относится к случаям, когда проектируемая БД содержит не слишком малое число таблиц. Скорее всего, можно обойтись без семантического моделирования, если число таблиц не превышает десяти, но оно становится совершенно необходимым, если БД включает более сотни таблиц. Для справедливости заметим, что ручная процедура создания концептуальной схемы с ее последующим преобразованием к реляционной схеме БД в случае больших БД (несколько сотен таблиц) затруднительна. Причины, по всей видимости, не требуют пояснений.

История систем автоматизации проектирования баз данных (CASE-средств) начиналась с автоматизации процесса рисования диаграмм, проверки их формальной корректности, обеспечения средств долговременного хранения диаграмм и другой проектной документации. Конечно, компьютерная поддержка работы с диаграммами очень полезна для проектировщика БД. Наличие электронного архива проектной документации помогает при эксплуатации, администрировании и сопровождении базы данных. Но система, которая ограничивается поддержкой рисования диаграмм, проверкой их корректности и хранением, напоминает текстовый редактор, поддерживающий ввод, редактирование и проверку синтаксической корректности конструкций некоторого языка программирования, но существующий отдельно от компилятора. Кажется естественным желание расширить такой редактор функциями компилятора, и это действительно возможно, поскольку известна техника компиляции конструкций языка программирования в коды целевого компьютера. Но коль скоро имеется четкая методика преобразования концептуальной схемы БД в реляционную схему, то почему бы ни выполнить

программную реализацию соответствующего "компилятора" и не включить ее в состав системы проектирования баз данных?

Эта простая мысль, естественно, не могла не придти в головы производителей CASE-средств проектирования БД. Подавляющее большинство подобных систем, представленных на рынке, обеспечивает автоматизированное преобразование диаграммных концептуальных схем баз данных, представленных в той или иной семантической модели данных, в реляционные схемы, представленные чаще всего на языке SQL. У читателя может возникнуть вопрос, почему в предыдущем предложении говорится про "автоматизированное", а не про "автоматическое" преобразование? Все дело в том, что в типичной схеме SQL-ориентированной БД могут содержаться определения многих объектов (ограничений целостности общего вида, триггеров и хранимых процедур и т.д.), которые невозможно сгенерировать автоматически на основе концептуальной схемы. Поэтому на завершающем этапе проектирования реляционной схемы снова требуется ручная работа проектировщика.

Еще раз обратите внимание на то, какой ход рассуждений привел нас к выводу о возможности автоматизации процесса преобразования концептуальной схемы БД в реляционную. Если создатели семантической модели данных предоставляют методику преобразования концептуальных схем в реляционные, достаточную для ее применения человеком, то почему бы ни реализовать программу, которая производит те же преобразования, следуя той же методике? Зададимся теперь другим, но, по существу, похожим вопросом. Если создатели семантической модели данных предоставляют язык (например, диаграммный), используя который проектировщики БД могут на основе исходной информации о предметной области сформировать концептуальную схему БД, то почему бы не реализовать программу, которая сама генерирует концептуальную схему БД в соответствующей семантической модели, используя исходную информацию о предметной области? Хотя мне неизвестны коммерческие CASE-средства проектирования БД, поддерживающие такой

подход, экспериментальные системы успешно существовали. Они представляли собой интегрированные системы проектирования с автоматизированным созданием концептуальной схемы на основе интервью с экспертами предметной области и последующим преобразованием концептуальной схемы в реляционную.

Существует много разных подходов к семантическому моделированию баз данных. В последние 10 лет одним из наиболее популярных языков семантического моделирования является UML. Проектирование реляционных БД - только одна и не слишком большая область применения этого языка, его возможности гораздо шире, однако подмножество UML (диаграммы классов) успешно применяется именно для таких целей.

Языковой механизм диаграмм классов по своей сути не отличается от существенно ранее внедренного в практику языкового механизма ER-диаграмм. Тем не менее, проектирование реляционных баз данных в среде UML дает одно существенное преимущество: можно выполнить весь проект создания информационной системы на основе одного общего инструмента.

Языку объектно-ориентированного моделирования UML (Unified Modeling Language) посвящено великое множество книг, многие из которых переведены на русский язык (а некоторые и написаны российскими авторами). UML разработан и развивается

Операции класса определяются в разделе, расположенном ниже раздела с атрибутами. При этом можно ограничиться только указанием имен операций, оставив более детальную спецификацию выполнения операций на поздние этапы моделирования. Для именования операций рекомендуется использовать глагольные обороты, соответствующие ожидаемому поведению объектов данного класса. Описание операции может также содержать ее сигнатуру, т.е. имена и типы всех параметров, а если операция является функцией, то и тип ее значения. Класс Человек с определенными операциями показан на рисунке 14.

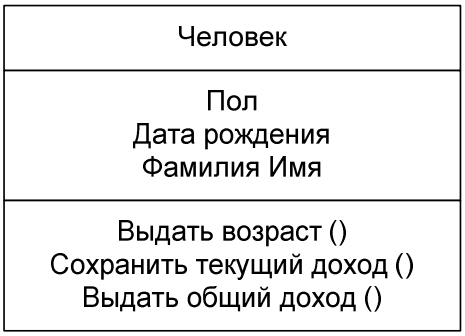

Рисунок 14 – Класс Человек

Для класса Человек мы определили три операции. Операция "выдатьВозраст" использует значение атрибута "датаРождения" и значение текущей даты. Операция "сохранитьТекущийДоход" позволяет зафиксировать в состоянии объекта сумму и дату поступления дохода данного человека. Операция "выдатьОбщийДоход" выдает суммарный доход данного человека за указанное время. Заметим, что состояние объекта меняется при выполнении только второй операции. Результаты первой и третьей операций формируется на основе текущего состояния объекта.

В диаграмме классов могут участвовать связи трех разных категорий: зависимость (dependency), обобщение (generalization) и ассоцирование (association). Для проектирования реляционных БД наиболее важны вторая и третья категории связей. Поэтому по поводу связей-зависимостей мы будет очень кратки.

Зависимостью называют связь по использованию, когда изменение в спецификации одного класса может повлиять на поведение другого, использующего первый класс. Чаще всего зависимости применяются в диаграммах классов, чтобы отразить в сигнатуре операции одного класса тот факт, что параметром этой операции могут быть объекты другого класса. Понятно, что если интерфейс этого второго класса изменяется, то это влияет на поведение объектов первого класса. Простой пример диаграммы классов со связью-зависимостью показан на рисунке 15.

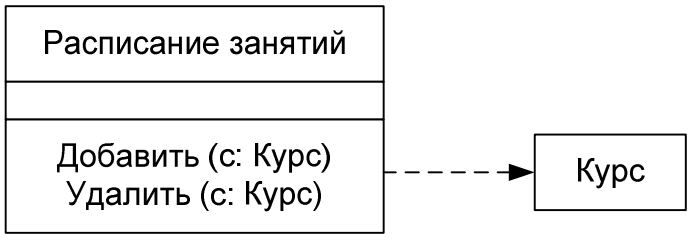

Рисунок 15 - Диаграмма классов

Зависимость показывается прерывистой линией со стрелкой, направленной к классу, от которого имеется зависимость. Легко видеть, что связи-зависимости существенны для объектно-ориентированных систем (в том числе и для ООБД).

Диаграммой классов в терминологии UML называется диаграмма, на которой показан набор классов (и некоторых других сущностей, не имеющих явного отношения к проектированию БД), а также связей между этими классами (иногда термин relationship переводится на русский язык не как "связь", а как "отношение"). Кроме того, диаграмма классов может включать комментарии и ограничения. Ограничения могут неформально задаваться на естественном языке или же могут формулироваться на языке OCL (Object Constraints Language), который является частью общей спецификации UML, но, в отличие от других частей языка, имеет не графическую, а линейную нотацию.

Классом называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой. Графически класс изображается в виде прямоугольника. У каждого класса должно иметься имя (текстовая строка), уникально отличающее его ото всех других классов. При формировании имен классов в UML допускается использование произвольной комбинации букв, цифр н даже знаков препинания. Однако на практике рекомендуется использовать в качестве имен классов короткие и осмысленные прилагательные и существительные, каждое из которых начинается с заглавной буквы. Примеры описания классов показаны на рисунке 16.

Рисунок 16 – Описание классов

Атрибутом класса называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов (в частности, не иметь ни одного атрибута). Свойство, выражаемое атрибутом, является свойством моделируемой сущности, общим для всех объектов данного класса. Так что атрибут является абстракцией состояния объекта. Любой атрибут любого объекта класса должен иметь некоторое значение (рисунок 17).

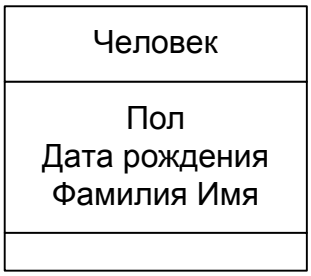

Рисунок 17 – Класс Человек с указанными именами атрибутов

Имена атрибутов представляются в разделе класса, расположенном под именем класса. Хотя UML не накладывает ограничений на способы формирования имен атрибутов (имя атрибута может быть произвольной текстовой строкой), на практике рекомендуется использовать короткие прилагательные и существительные, смысл которых соответствует смыслу соответствующего свойства класса. Первое слово в имени атрибута рекомендуется писать с прописной буквы, а все остальные слова - с заглавной буквы.

Операцией класса называется именованная услуга, которую можно запросить у любого объекта этого класса. Операция - это абстракция того, что можно делать с объектом. Класс может содержать любое число операций (в

частности, не содержать ни одной операции). Набор операций класса является общим для всех объектов данного класса.

Связью-обобщением называется связь между общей сущностью, называемой суперклассом, или родителем, и более специализированной разновидностью этой сущности, называемой подклассом, или потомком. Обобщения иногда называют связями "is a", имея в виду, что класс-потомок является частным случаем класса-предка. Класс-потомок наследует все атрибуты и операции класса-предка, но в нем могут быть определены дополнительные атрибуты и операции.

Объекты класса-потомка могут использоваться везде, где могут использоваться объекты класса-предка. Это свойство называют полиморфизмом по включению, имея в виду, что объекты потомка можно считать включаемыми в класс-предок. Графически обобщения изображаются в виде сплошной линии с большой незакрашенной стрелкой, направленной к суперклассу. На рисунке 18 показан пример иерархии одиночного наследования: у каждого подкласса имеется только один суперкласс.

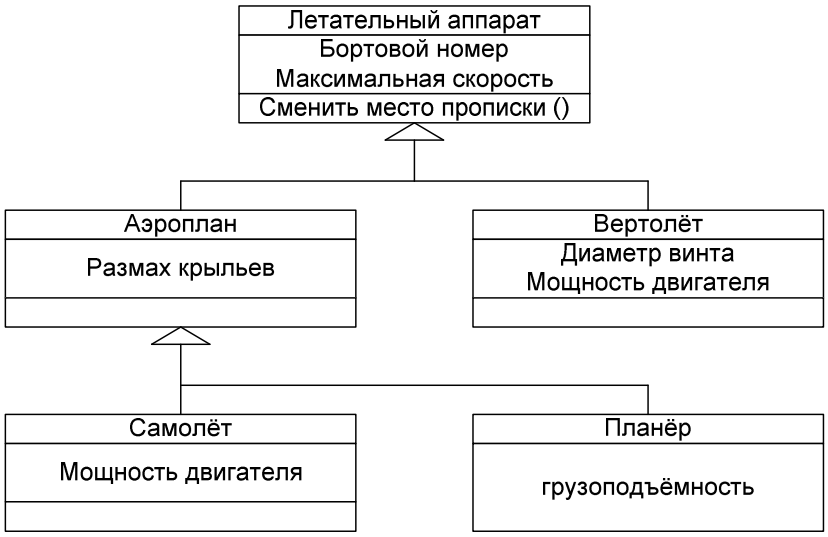

Рисунок 18 – Иерархия одиночного наследования классов

Одиночное наследование является достаточным в большинстве практических случаев применения связи-обобщения. Однако в UML допускается и множественное наследование, когда один подкласс определяется

на основе нескольких суперклассов (рисунок 19). В качестве одного из разумных примеров рассмотрим следующую диаграмму классов (для упрощения диаграммы не будем указывать имена атрибутов и операций).

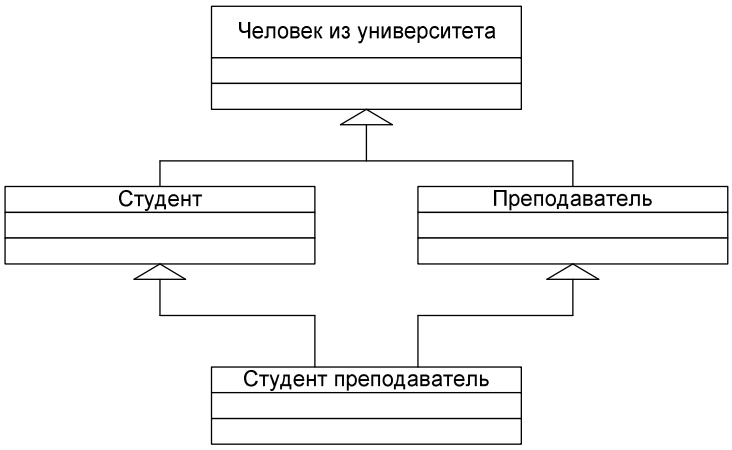

Рисунок 19 – Пример множественного наследования классов

На этой диаграмме классы Студент и Преподаватель порождены из одного суперкласса ЧеловекИзУниверситета. Но как это часто случается, многие студенты уже в студенческие годы начинают преподавать, так что могут существовать такие два объекта классов Студент и Преподаватель, которым соответствует один объект класса ЧеловекИзУниверситета. Итак, среди объектов класса Студент могут быть преподаватели, а некоторые преподаватели могут быть студентами. Мы можем определить класс СтудентПреподаватель путем множественного наследования от суперклассов Студент и Преподаватель. Объект класса СтудентПреподаватель обладает всеми свойствами и операциями классов Студент и Преподаватель, и может быть использован везде, где могут использоваться объекты этих классов. Так что полиморфизм по включению продолжает работать.

Но множественное наследование, помимо того, что не слишком часто требуется на практике, порождает ряд проблем, из которых одной из наиболее известных является проблема именования атрибутов и операций в подклассе, полученном путем множественного наследования. Например, предположим, что при образовании подклассов Студент и Преподаватель в них обоих был

создан атрибут с именем "номерКомнаты". Очень вероятно, что для объектов класса Студент значениями этого атрибута будут номера комнат в студенческом общежитии, а для объектов класса Преподаватель - номера служебных кабинетов. Как быть с объектами класса СтудентПреподаватель, для которых существенны оба одноименных атрибута? На практике применяется одно из следующих решений:

1. запретить образование подкласса СтудентПреподаватель, пока в одном из суперклассов не будет произведено переименование атрибута "номерКомнаты";

2. наследовать это свойство только от одного из суперклассов, так что, например, значением атрибута "номерКомнаты" у объектов класса СтудентПреподаватель всегда будут номера служебных кабинетов;

3. унаследовать в подклассе оба свойства, но автоматически переименовать оба атрибута, чтобы прояснить их смысл; назвать их, например, "номерКомнатыСтудента" и "номерКомнатыПреподавателя".

Ни одно из решений не является полностью удовлетворительным. Первое решение требует возврата к ранее определенному классу, имена атрибутов и операций которого, возможно, уже используются в приложениях. Второе решение нарушает логику наследования, не давая возможности на уровне подкласса использовать все свойства суперклассов. Наконец, третье решение заставляет использовать длинные имена атрибутов и операций, которые могут стать недопустимо длинными, если процесс множественного наследования будет продолжаться от полученного подкласса.

Но, конечно, сложность проблемы именования атрибутов и операций не идет ни в какое сравнение со сложностью реализации множественного наследования в реляционных БД. Поэтому при использовании UML для проектирования реляционных БД нужно очень осторожно использовать наследование классов и стараться вообще избегать множественного наследования.

Ассоциацией называется структурная связь, показывающая, что объекты одного класса некоторым образом связаны с объектами другого или того же самого класса. Допускается, чтобы оба конца ассоциации относились к одному классу. В ассоциации могут связываться два класса, и тогда она называется бинарной. Допускается создание ассоциаций, связывающих сразу n классов (они называются n-арными ассоциациями). Графически ассоциация изображается в виде линии, соединяющей класс сам с собой или с другими классами.

С понятием ассоциации связаны четыре важных дополнительных понятия: имя, роль, кратность и агрегация. Во-первых, ассоциации может быть присвоено имя, характеризующее природу связи. Смысл имени уточняется черным треугольником, располагаемым над линией связи справа или слева от имени ассоциации. Этот треугольник указывает направление, в котором должно читаться имя связи. Пример именованной ассоциации показан на рисунке 20. Треугольник означает, что именованная ассоциация должна читаться, как "Студент учится в Университете".

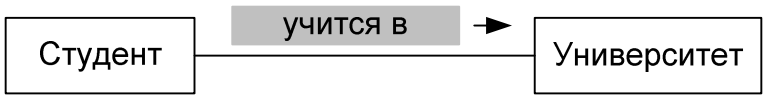

Рисунок 20 – Пример именованной ассоциации

Другим способом именования ассоциации является указание роли каждого класса, участвующего в этой ассоциации. Роль класса задается именем, помещаемым под линией ассоциации ближе к данному классу. На следующем рисунке показаны две ассоциации между классами Человек и Университет, в которых эти классы играют разные роли. Как мы видим, объекты класса Человек могут выступать в роли РАБОТНИКОВ при участии в ассоциации, в которой объекты класса Университет играют роль НАНИМАТЕЛЯ. В другой ассоциации объекты класса Человек играют роль СТУДЕНТА, а объекты класса УНИВЕРСИТЕТ - в роли ОБУЧАЮЩЕГО (рисунок 21).

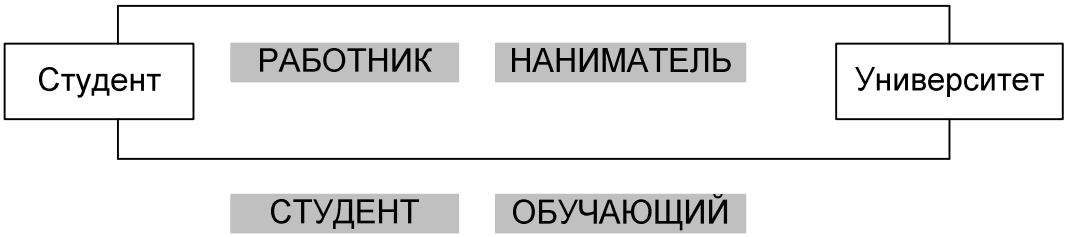

Рисунок 21 – Две ассоциации с разными ролями классов

В общем случае, для ассоциации может задаваться и ее собственное имя, и имена ролей классов. Это связано с тем, что класс может играть одну и ту же роль в разных ассоциациях, так что в общем случае пара имен ролей классов не идентифицирует ассоциацию. С другой стороны, в простых случаях, когда между двумя классами определяется только одна ассоциация, можно вообще не связывать с ней дополнительные имена.

Кратностью (multiplicity) роли ассоциации называется характеристика, указывающая, сколько объектов класса с данной ролью может или должно участвовать в каждом экземпляре ассоциации (в UML экземпляр ассоциации называется соединением - link). Наиболее распространенным способом задания кратности роли ассоциации является указание конкретного числа или диапазона. Например, указание "1" говорит о том, что все объекты класса с данной ролью должны участвовать в некотором экземпляре данной ассоциации, причем в каждом экземпляре ассоциации может участвовать ровно один объект класса с данной ролью. Указание диапазона "0..1" говорит о том, что не все объекты класса с данной ролью обязаны участвовать в каком-либо экземпляре данной ассоциации, но в каждом экземпляре ассоциации может участвовать только один объект. Аналогично, указание диапазона "1..*" говорит, что все объекты класса с данной ролью должны участвовать в некотором экземпляре данной ассоциации, и в каждом экземпляре ассоциации должен участвовать хотя бы один объект (верхняя граница не задана). Толкование диапазона "0..*" является очевидным расширением случая "0..1".

В более сложных (но крайне редко встречающихся на практике) случаях определения кратности можно использовать списки диапазонов. Например, список "2, 4..6, 8..*" говорит, что все объекты класса с указанной ролью должны участвовать в некотором экземпляре данной ассоциации, и в каждом экземпляре ассоциации должны участвовать два, от четырех до шести или более восьми объектов класса с данной ролью (рисунок 22).

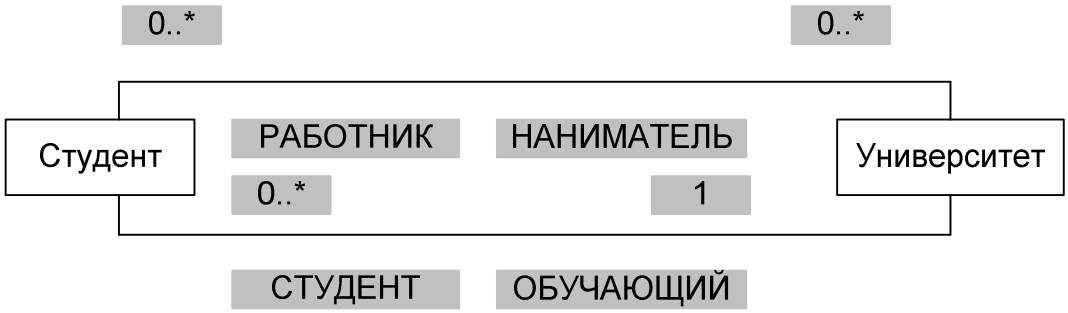

Рисунок 22 – Ассоциация с указанными кратностями ролей

На диаграмме классов с рисунка показано, что произвольное (может быть, нулевое) число людей являются работниками произвольного числа университетов. Каждый университет обучает произвольное (может быть, нулевое) число студентов, но каждый студент может быть студентом только одного университета.

Обычная ассоциация между двумя классами характеризует связь между равноправными сущностями: оба класса находятся на одном концептуальном уровне. Иногда в диаграмме классов требуется отразить тот факт, что ассоциация между двумя классами имеет специальный вид "часть-целое". В этом случае класс "целое" имеет более высокий концептуальный уровень, чем класс "часть". Ассоциация такого рода называется агрегатной. Графически агрегатные ассоциации изображаются в виде простой ассоциации с незакрашенным ромбом на стороне класса-"целого". Простой пример агрегатной ассоциации показан на рисунке 23.

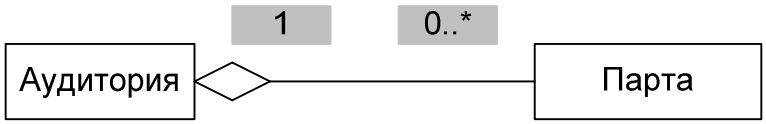

Рисунок 23 – Пример агрегатной ассоциации

Объектами класса Аудитория являются студенческие аудитории, в которых проходят занятия. В каждой аудитории должны быть установлены парты. Поэтому в некотором смысле класс Парта является "частью" класса Аудитория. Мы умышленно сделали роль класса Парта необязательной, поскольку могут существовать аудитории без парт (например, класс для занятий танцами), и некоторые парты могут находиться на складе. Обратите внимание, что хотя аудитории, не оснащенные партами, как правило, непригодны для занятий, объекты классов Аудитория и Парта существуют независимо. Если некоторая аудитория ликвидируется, то находящиеся в ней парты не уничтожаются, а переносятся на склад.

Бывают случаи, когда связь "части" и "целого" настолько сильна, что уничтожение "целого" приводит к уничтожению всех его "частей". Агрегатные ассоциации, обладающие таким свойством, называются композитными, или просто композициями. При наличии композиции объект-часть может быть частью только одного объекта-целого (композита). При обычной агрегатной ассоциации "часть" может одновременно принадлежать нескольким "целым". Графически композиция изображается в виде простой ассоциации, дополненной закрашенным ромбом со стороны "целого". Пример композитной агрегатной ассоциации показан на рисунке 24.

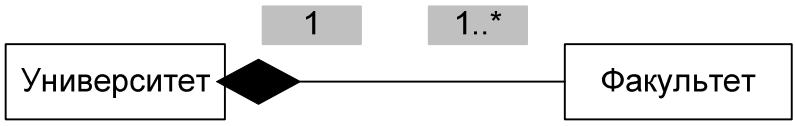

Рисунок 24 – Пример композитной агрегатной ассоциации

Любой факультет является частью одного университета, и ликвидация университета приводит к ликвидации всех существующих в нем факультетов

(хотя во время существования университета отдельные факультеты могут ликвидироваться и создаваться).

Заметим, что в контексте проектирования реляционных БД агрегатные и в особенности композитные ассоциации влияют только на способ поддержки ссылочной целостности. В частности, композитная связь является явным указанием того, что ссылочная целостность между "целым" и "частями" должна поддерживаться путем каскадного удаления частей при удалении целого. При наличии простой ассоциации между двумя классами предполагается возможность навигации между объектами, входящими в один экземпляр ассоциации. Если известен конкретный объект-студент, то должна обеспечиваться возможность узнать соответствующий объект-университет. Если известен конкретный объект-университет, то должна обеспечиваться возможность узнать все соответствующие объекты-студенты. Другими словами, если не оговорено противное, то навигация по ассоциации может проводиться в обоих направлениях. Однако бывают случаи, в которых желательно ограничить направление навигации для некоторых ассоциаций. В этом случае на линии ассоциации ставится стрелка, указывающая направление навигации. Возможный пример показан на рисунке 25.

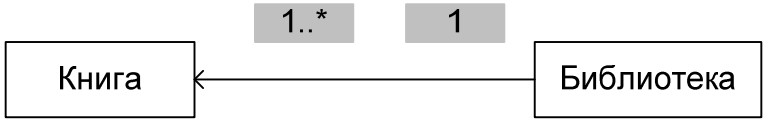

Рисунок 25 – Ассоциация с указанным направлением навигации

В библиотеке должно содержаться некоторое число книг, но каждая книга должна принадлежать некоторой библиотеке. С точки зрения библиотечного хозяйства разумно иметь возможность найти книгу в библиотеке, т.е. произвести навигацию от объекта-библиотеке к связанным с ним объектам-книгам. Однако вряд ли потребуется по данному экземпляру книги узнать, в какой библиотеке она находится.

Как уже отмечалось, в диаграммах классов могут указываться ограничения целостности, которые должны поддерживаться в проектируемой БД. Имеются два способа определения ограничений: на естественном языке и на языке OCL. На рисунке 26 показана простая диаграмма классов Студент и Университет с ограничением, выраженном на естественном языке.

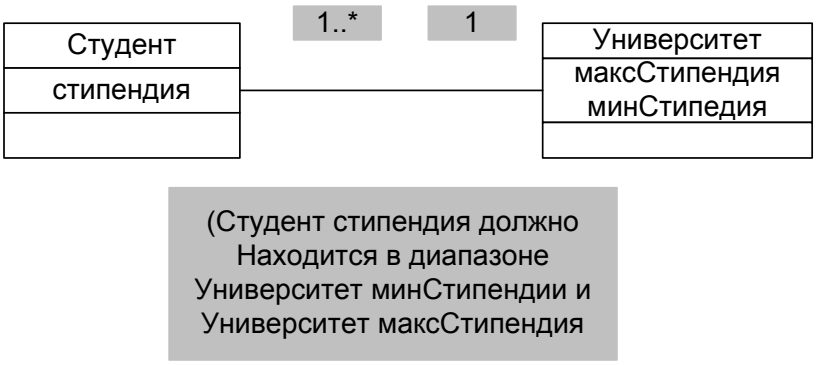

Рисунок 26 – Ограничение, выраженное на естественном языке

В данном случае накладывается ограничение на состояние объектов классов Студент и Университет, входящих в один экземпляр ассоциации. Объект класса Студент может входить в экземпляр связи с объектом класса Университет только при том условии, что размер стипендии данного студента находится в диапазоне, допустимом в данном университете.

Более точный и лаконичный способ формулировки ограничений обеспечивает язык OCL (Object Constraint Language). Приведем его сжатое описание.

К заимствованным из UML концепциям относятся, в первую очередь, следующие:

класс, операция, атрибут;

объект (экземпляр класса);

ассоциация;

тип данных (включая набор предопределенных типов Boolean, Integer, Real и String);

значение (экземпляр типа данных).

Существенными для понимания языка OCL являются определенные в UML отличия между объектом некоторого класса и значением некоторого типа, обычные для объектных моделей данных.

объект имеет уникальный идентификатор и может сравниваться с другими объектам только по значению идентификатора, следствием чего является возможность определения множественных операций над объектами в терминах их идентификаторов;

объект может быть ассоциирован через бинарную связь с другими объектами, что позволяет определить в OCL операцию перехода от объекта к связанным с ним объектам.

Вообще говоря, переход от диаграммного представления концептуальной схемы базы данных к ее реляционной схеме не зависит от разновидности используемых диаграмм. В частности, методика, разработанная для классических диаграмм «Сущность-Связь» (Entity-Relationship), практически всегда пригодна для диаграмм классов UML.

Выводы: 1. Прежде, чем определять в классах операции, подумайте, что можно сделать с этими определениями в среде целевой РСУБД. Если в этой среде поддерживаются хранимые процедуры, то, возможно, некоторые операции могут быть реализованы именно с помощью такого механизма. Но если в среде РСУБД поддерживается механизм определяемых пользователями функций, он может оказаться более подходящим.

2. Помните, что сравнительно эффективно в РСУБД реализуются только ассоциации видов "один-ко-многим" и "многие-ко-многим". Если в созданной диаграмме классов имеются ассоциации "один-к-одному", то следует задуматься о целесообразности такого проектного решения. Реализация в среде РСУБД ассоциаций с точно заданными кратностями ролей возможна, но требует определения дополнительных триггеров, выполнение которых понизит эффективность.

3. Для технологии реляционных БД агрегатные и в особенности композитные ассоциации являются неестественными.

4. В спецификации UML говорится, что, определяя однонаправленные связи, можно способствовать получению эффективности доступа к некоторым объектам. Для технологии реляционных баз данных поддержка такого объявления вызовет дополнительные накладные расходы и тем самым снизит эффективность.

5. Не злоупотребляйте возможностями OCL. Диаграммы классов UML - это хороший и мощный инструмент для создания концептуальных схем баз данных.

Нельзя сказать, что проектирование баз данных на основе семантических моделей в любом случае убыстряет и/или упрощает процесс проектирования. Все зависит от сложности предметной области, квалификации проектировщика и качества вспомогательных программных средств. Но в любом случае этап диаграммного моделирования обеспечивает следующие преимущества:

на раннем этапе проектирования до привязки к конкретной РСУБД проектировщик может обнаружить и исправить логические огрехи проекта, руководствуясь наглядным графическим представлением концептуальной схемы;

окончательный вид концептуальной схемы, полученной непосредственно перед переходом к формированию реляционной схемы, а может быть, и промежуточные версии концептуальной схемы должны стать частью документации целевой реляционной БД; наличие этой документации очень полезно для сопровождения и в особенности для изменения схемы БД в связи с изменившимися требованиями;

при использовании CASE-средств концептуальное моделирование БД может стать частью всего процесса проектирования целевой информационной системы, что может способствовать правильной структуризации процесса, эффективности и повышению качества проекта в целом.

Язык UML принадлежит объектному миру. Этот мир гораздо сложнее реляционного мира. Поскольку UML может использоваться для унифицированного объектно-ориентированного моделирования всего, что

угодно, в нем содержится масса различных понятий, терминов и вариантов использования, совершенно избыточных с точки зрения проектирования реляционных БД.

Домашнее задание. Построить концептуальную модель базы данных для выбранного объекта управления.

Работа 10
РАСЧЁТНО-ГРАФИЧЕСКАЯ РАБОТА

Изучение работ по объектно-ориентированному методу анализа и проектированию должна завершатся самостоятельной работой , объединяющей все домашние заголовки в общую расчётно-графическую работу.

Состав расчётно-графической работы: аннотация, введение, диаграмма процесса управления, циклограмма, модель управления, сеть Петри, дерево достижимости, концептуальная модель системы управления на языке UML, заключение и список использованных источников.

В заключении необходимо сделать выводы о дальнейшей работе (дипломной, курсовой). Особо отметить вопросы патентных проработок, а лучше приложить проект заявки на предполагаемое изобретение по результатам анализа и проектирования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Шемелин. В.А. Проектирование систем управления в машиностроении. Учебник для ВУЗов. М., 1998 - 254 с.

2. Вендров А.М. CASE – технологии. Современные методы и средства проектирования информационных систем. М., Финансы и статистика, 1998.

3. Вендров А.М. Проектирование программного обеспечения экономических информационных систем. М., Финансы и статистика, 2000.

В книгах А.М. Вендрова даётся широкий обзор современных технологий проектирования информационных систем. Автор руководствуется собственным практическим опытом, и в его изложении сглаживаются терминологические и концессионные барьеры между разными подходами. Кроме того, изначально эти книги написаны на русском языке, а не переведены с английского, поэтому читаются без затруднений.

4. Фаулер М., Скотт К. UML в кратком изложении. Применение стандартного языка объектного моделирования. М., Мир, 1999.

Многие мои коллеги считают, что это лучшая книга про UML, переведённая на русский язык. Она написана чётко, без повторов и философских отклонений от темы. Конечно, книга не может улучшить язык, но она помогает понять его в том виде, в коком он существует. Кстати, переводил книгу А.М. Вендров.

5. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. 2-е изд. М., Издательство Бином, СПб., Невский диалект, 1999.

6. Буч Г., Рамбо Д., Джекобсон А. Язык UML: руководство пользователя. М., ДМК, 2000.

7. Кудинов А.А., Серов А.Е. Проектирование систем автоматизации. Учебное пособие по курсовому и дипломному проектированию с грифом ДВРУМЦ, - Благовещенск: Амурск ГОС.УН-Т, 2002 - 124 с.

СОДЕРЖАНИЕ

А.А. Кудинов. Проектирование автоматизированных систем: методические указания к лабораторно-практическим занятиям по объектно-ориентированным методам проектирования. – Благовещенск, издательство АмГУ, 2010, 81с.