ГОСУДАРСТВЕННОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Экономика и управление»

АВТОМАТИЗАЦИЯ УПРАВЛЕНЧЕСКОЙ ДЕЯТЕЛЬНОСТИ

Методические рекомендации к контролируемой самостоятельной работе для студентов направления подготовки 27.03.05 «Инноватика» дневной формы обучения

Могилёв 2017

УДК 338.24:004
ББК 65.290-2:32.97
А 22

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Экономика и управление» «28» сентября 2017 г., протокол № 2

Составитель канд. экон. наук, доц. Е. С. Жесткова

Рецензент канд. экон. наук, доц. Н. С. Желток

Приведены задания к контролируемой самостоятельной работе по дисциплине «Автоматизация управленческой деятельности» и даны рекомендации по их выполнению.

Учебно-методическое издание

АВТОМАТИЗАЦИЯ УПРАВЛЕНЧЕСКОЙ ДЕЯТЕЛЬНОСТИ

Ответственный за выпуск	Т. В. Пузанова
Технический редактор	А. Т. Червинская
Компьютерная вёрстка	Е. С. Лустенкова

Подписано в печать . Формат 60×84 /16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 36 экз. Заказ №

Издатель и полиграфическое исполнение:
Государственное учреждение высшего профессионального образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 24.01.2014.
Пр. Мира, 43, 212000, Могилёв.

Содержание

Введение

В современных условиях работа предприятий и организаций немыслима без применения программного обеспечения, облегчающего обработку экономической информации в процессе принятия управленческих решений и реализации управленческой деятельности. Зачастую требуемое программное обеспечение создаётся силами сотрудников предприятия. Успех подобных разработок напрямую зависит от способности проектировщика и разработчика предварить их создание и внедрение описанием всего комплекса проблем, которые необходимо разрешить, указанием того, какие функции системы должны быть автоматизированы, как взаимодействует система со своим окружением. Иными словами, этап проектирования является критическим для разработки высококачественного программного обеспечения.

Центральным элементом процесса проектирования программного обеспечения выступает моделирование. Модели позволяют наглядно продемонстрировать желаемую структуру и поведение системы. Они дают возможность рассмотреть проектируемую систему с разных точек зрения, что помогает добиться лучшего понимания создаваемой программной системы, что зачастую приводит к её упрощению и возможности повторного использования. Наконец, модели необходимы для минимизации риска при проектировании программного обеспечения.

От моделирования может выиграть любой проект. Даже при создании одноразовых программ моделирование помогает разработчику лучше представить план системы, а значит, выполнить проект быстрее и представить именно то, что подразумевал изначальный замысел. Чем сложнее проект, тем более вероятно, что из-за отсутствия моделирования он потерпит неудачу или будет получена не та система, которая требовалась. Кроме того, все полезные и интересные системы с течением времени обычно усложняются. Отказ от моделирования в самом начале работы над системой приводит к затруднениям, устранить которые на более поздних этапах бывает практически невозможно.

Изучение способов и инструментов моделирования, применяемых на различных этапах создания программного обеспечения, является целью при выполнении контрольной работы по дисциплине «Автоматизация управленческой деятельности».

1 Цель и задачи контрольной работы

Предусмотренная рабочей программой дисциплины «Автоматизация управленческой деятельности» контролируемая самостоятельная работа студентов заключается в разработке проекта программного обеспечения, автоматизирующего одну из задач, возникающих в практической деятельности организации.

Целью контрольной работы является закрепление теоретических навыков по курсу «Автоматизация управленческой деятельности», а также развитие навыков самостоятельной творческой работы студентов для дипломного проектирования.

Для достижения указанной цели в контрольной работе решаются следующие задачи:

- проведение анализа потоков информации и структуры исследуемого объекта;
- проектирование программного обеспечения.

В ходе выполнения контрольной работы студенты формируют описание проектируемой системы, объектно-ориентированные, функциональные и информационные модели с использованием инструментальных систем поддержки проектирования программного обеспечения.

2 Задание

Задание на контрольную работу выдаётся преподавателем. Оно включает в себя тему работы, исходные данные, перечень подлежащих разработке вопросов.

3 Содержание контрольной работы

Объём работы зависит от сложности разрабатываемого программного обеспечения, но не должен превышать 15–20 страниц текста с необходимыми таблицами, рисунками и приложениями.

Контрольная работа должна включать:

- титульный лист;
- задание;
- содержание;
- введение;
- расчётную часть;
- заключение;
- литературу;
- приложения.

4 Указания к выполнению контрольной работы

Успешное выполнение контрольной работы возможно при соблюдении определённой последовательности. Далее предлагается следующий порядок работы над заданием.

Этап I. Подбор и изучение литературы по теме, анализ исходной информации (исследование бизнес-процессов, протекающих при решении выбранной задачи экономической деятельности). Построение и анализ схемы бизнес-процессов.

Этап II. Определение основных требований к программному обеспечению. Построение и описание UML-диаграмм, отражающих статические и динамические аспекты проектируемой программной системы. Разработка пользовательского интерфейса, основных алгоритмов, видов представления входной и выходной информации.

Этап III. Оформление контрольной работы.

Во введении необходимо указать место программных систем при решении задач автоматизации управленческой деятельности, актуальность их применения на предприятиях Республики Беларусь и Российской Федерации.

В первом разделе следует подробно описать специфику выбранной задачи, а также провести анализ требований, которым должна удовлетворять программная система. При объектно-ориентированном подходе анализ требований сводится к разработке моделей бизнес-процессов системы. Модели, созданные и отлаженные в начале разработки, используются на последующих этапах, облегчая программирование системы, её отладку и тестирование, сопровождение и дальнейшую модификацию.

Для более наглядного представления данной информации необходимо построить модель бизнес-процессов отдела с использованием стандартов IDEF0.

Во втором разделе контрольной работы производится проектирование программной системы с использованием универсального языка моделирования UML. В этом разделе следует подробно описать архитектуру проектируемой системы с разных точек зрения, выявить дополнительные требования, создать макет программной системы.

В заключении даётся краткая характеристика проделанной работы, а также отмечается её значимость при решении задач автоматизации управленческой деятельности.

4.1 Системный анализ и анализ требований к программной системе

Проектирование программной системы начинается с анализа требований, которым она должна удовлетворять. Как правило, системный анализ и анализ требований сводятся к разработке моделей системы.

Для того чтобы провести анализ требований, нужно подробно описать специфику задачи, выбранной для автоматизации. В частности, необходимо собрать следующую информацию:

– отдел, в котором возникла задача, его основные цели и функции;

– состав отдела с указанием количества специалистов каждой квалификации;

– работы, выполняемые сотрудниками отдела, в частности:

а) название работы;

б) входная информация, необходимая для выполнения работы (откуда приходит, в каком виде представлена, с какой частотой);

в) кто из специалистов выполняет данную работу;

г) какие технические и программные средства используются для выполнения работы;

д) выходная информация, получаемая в результате выполнения работы (куда направляется, в каком виде, с какой частотой);

– оснащённость отдела средствами автоматизации (наличие ПК, ПО, ЛВС).

Для более наглядного представления данной информации необходимо построить модель бизнес-процессов. Наиболее удобным языком моделирования бизнес-процессов является IDEF0, в соответствии с которым система представляется как совокупность взаимодействующих работ или функций. Функциональная ориентация является принципиальной – функции системы анализируются независимо от объектов, которыми они оперируют. Это позволяет чётко смоделировать логику и взаимодействие процессов организации.

Моделирование в IDEF0 начинается с создания контекстной диаграммы – диаграммы абстрактного уровня описания системы в целом, содержащей определение субъекта моделирования, цели и точки зрения на модель.

Под субъектом понимается сама система; на определение субъекта влияют позиция, с которой рассматривается система, и цель моделирования – вопросы, на которые построенная модель должна дать ответ.

Сначала определяется область моделирования. В ходе моделирования область может корректироваться, но изначально она должна быть сформулирована, поскольку задаёт направление моделирования. При формулировании области необходимо учитывать ширину и глубину моделирования. Ширина определяет, что будет рассматриваться внутри системы, а что снаружи; глубина – на каком уровне детализации модель является завершённой. При задании глубины системы необходимо помнить, что трудоёмкость построения модели растёт с увеличением глубины декомпозиции. После установления границ модели предполагается, что новые объекты не должны вноситься в моделируемую систему.

Также перед началом моделирования необходимо чётко определить цель проектируемой системы и позицию (точку зрения), с которой наблюдается система и создаётся модель. Цель моделирования состоит из ответов на вопросы: «Почему процесс должен быть смоделирован?», «Что должна показывать модель?», «Что может получить клиент?».

Под точкой зрения понимается перспектива, с которой наблюдалась система при построении модели. При построении модели учитываются мнения

различных людей, но все они должны придерживаться единой точки зрения на модель. Точка зрения должна соответствовать цели и границам моделирования. Как правило, выбирается точка зрения человека, ответственного за моделируемую работу в целом. Методология SADT требует, чтобы модель всегда рассматривалась с одной позиции. Точка зрения диктует выбор нужной информации и форму её подачи.

IDEF0-модель предполагает наличие чётко сформулированной цели, единственного субъекта моделирования и одной точки зрения. Конечным результатом является набор взаимоувязанных описаний, начиная с описания самого верхнего уровня всей системы и заканчивая подробным описанием деталей или операций системы. Такие описания называются диаграммами. IDEF0-модель объединяет и организует диаграммы в иерархические структуры, в которых диаграммы на верхнем уровне модели являются более общими, чем детализированные диаграммы нижних уровней. Каждая диаграмма выступает единицей описания системы и располагается на отдельном листе.

Модель может содержать следующие типы диаграмм:

- контекстную диаграмму (в модели она может быть лишь одна);
- диаграммы декомпозиции.

Контекстная диаграмма представляет собой общее описание системы и её взаимодействия с внешней средой. После описания системы в целом проводится разбиение её на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент, – диаграммами декомпозиции. После декомпозиции контекстной диаграммы проводится декомпозиция каждого большого фрагмента системы на более мелкие до достижения нужного уровня подробности описания. После каждого сеанса декомпозиции проводятся сеансы экспертизы – эксперты указывают на соответствие реальных бизнес-процессов созданным диаграммам. После прохождения экспертизы без замечаний можно приступать к следующему сеансу декомпозиции. Так достигается соответствие модели реальным бизнес-процессам на любом уровне модели.

Каждая IDEF0-диаграмма содержит блоки и дуги. Блоки соответствуют работам и обозначают процессы, функции или задачи, которые происходят в течение определённого времени и имеют распознаваемые результаты. Работы изображаются в виде прямоугольников. Все работы должны быть названы и определены. Имя работы выражается глаголом или отглагольным существительным, обозначающим действие (например, «Принять заказ», «Составление отчёта» и т. д.).

Стандарт IDEF0 требует, чтобы в диаграмме было не более шести блоков. Это обеспечивает удобство чтения и понимания диаграмм. Каждая сторона блока имеет определённое назначение. Левая сторона блока предназначена для входов, верхняя – для управления, правая – для выходов, нижняя – для механизмов. Такое обозначение отражает определённые системные принципы: входы преобразуются в выходы, управление оговаривает условия выполнения преобразований, механизмы показывают, кто, что и как выполняет функцию.

Дуги на диаграмме IDEF0 изображаются одинарными линиями со

стрелками на концах. Стрелки описывают взаимодействие работ, представляют собой некий объект или информацию и выражаются существительными (например, «Звонки клиентов», «Правила», «Бухгалтерская система»).

Между объектами и функциями возможны четыре отношения: вход, управление, выход, механизм. Каждое из них изображается дугой, связанной с определённой стороной блока.

Вход – материал или информация, которые используются или преобразуются работой для получения результата (выхода). Стрелка входа входит в левую сторону блока. Работа может не иметь стрелок входа. При описании технологических процессов не возникает проблем определения входов. При моделировании процессов обработки информации, когда стрелками являются данные, всё не так очевидно. Например, при выполнении функции «Приём пациента» карта пациента может быть на входе и на выходе, но качество этих данных меняется. Для того чтобы оправдать своё назначение, стрелки входа и выхода должны быть точно определены и указывать на то, что данные были переработаны (например, на выходе – «Заполненная карта пациента»). Часто сложно определить, являются ли данные входом или управлением. В этом случае помогает информация о том, изменяются данные в работе или нет. Если изменяются, то, скорее всего, это вход, если нет – управление.

Управление – правила, стратегии, процедуры или стандарты, которыми руководствуется работа. Каждая работа должна иметь хотя бы одну стрелку управления. Стрелка управления входит в верхнюю сторону блока. Управление влияет на работу, но не преобразуется ею. Если цель работы – изменить процедуру или стратегию, то эта процедура или стратегия будет для работы входом. В случае возникновения неопределённости в статусе стрелки (управление или вход) рекомендуется рисовать стрелку управления.

Выход – материал или информация, которые производятся работой. Каждая работа должна иметь хотя бы одну стрелку выхода. Работа без результата не имеет смысла и не должна моделироваться. Стрелка выхода исходит из правой стороны блока.

Механизм – ресурсы, которые выполняют работу, например, персонал предприятия, станки, ПО и т. д. Стрелка механизма входит в нижнюю сторону блока. По усмотрению механизмы могут не изображаться в модели.

Таким образом, IDEF0-диаграммы представляют входные-выходные преобразования и указывают правила этих преобразований. Дуги на них изображают интерфейсы между функциями системы, а также между системой и окружающей средой.

Стрелки на контекстной диаграмме служат для описания взаимодействия системы с окружающим миром. Они могут начинаться у границы диаграммы и заканчиваться у работы, или наоборот. Такие стрелки называются граничными. Пример контекстной диаграммы представлен на рисунке 1.

Диаграммы декомпозиции предназначены для детализации работы, следовательно, они содержат дочерние работы, имеющие общую родительскую работу. Нужно понимать, что работы нижнего уровня – это то же самое, что

работы верхнего уровня, но в более детальном изложении. Следовательно, границы работы верхнего уровня – то же самое, что границы диаграммы декомпозиции. При декомпозиции работы входящие в неё и исходящие из неё стрелки должны обязательно присутствовать на диаграмме декомпозиции.

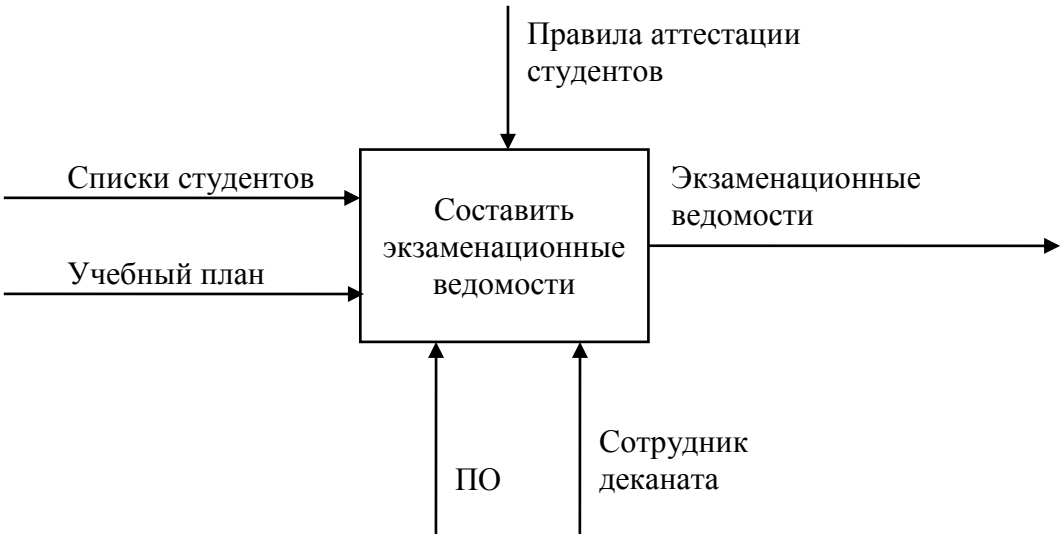


Рисунок 1 – Пример контекстной диаграммы

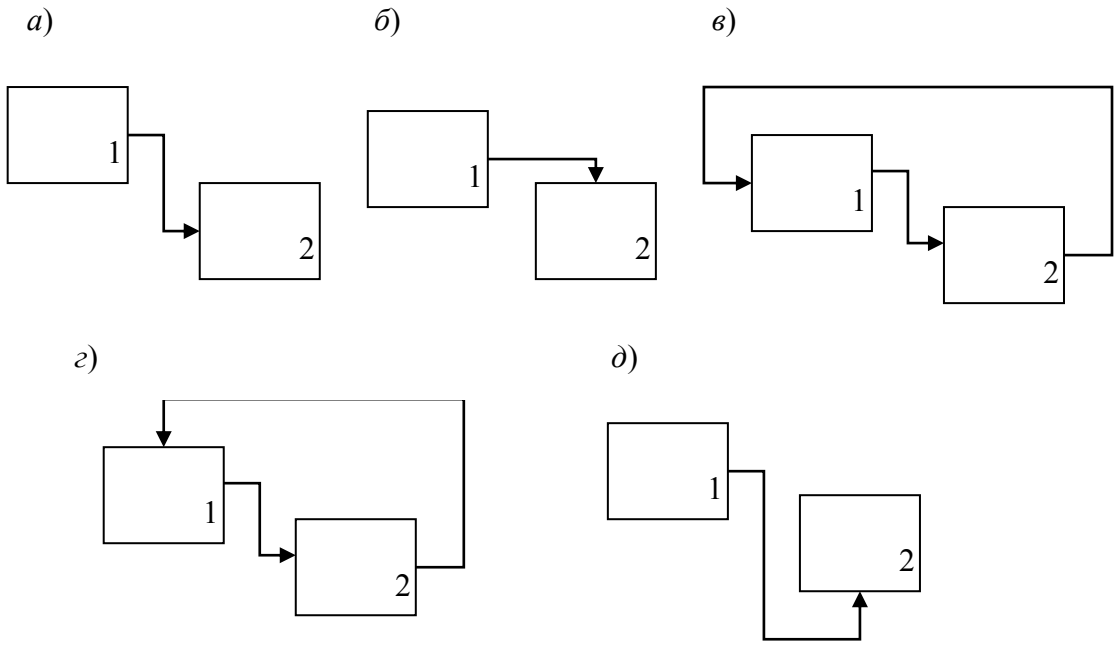
Допустимый интервал числа дочерних работ на диаграмме – от двух до восьми. Декомпозировать работу на одну работу не имеет смысла, а диаграммы с количеством работ более восьми плохо читаются. Для обеспечения наглядности и понимания моделируемых процессов рекомендуется использовать от трёх до шести блоков на одной диаграмме. Каждая из работ на диаграмме может быть декомпозирована. В левом верхнем углу блока изображается небольшая диагональная черта, которая показывает, что данная работа не была декомпозирована.

Работы на диаграммах располагаются по диагонали от левого верхнего угла к правому нижнему. Такой порядок называется порядком доминирования. Доминирование – это влияние, которое один блок оказывает на другие блоки диаграммы. В левом верхнем углу обычно помещается самая важная или выполняемая по времени первой работа. Вправо вниз располагаются менее важные или выполняемые позже работы. Такое размещение облегчает чтение диаграмм, кроме того, на нём основывается понятие взаимосвязей работ.

Блоки в IDEF0 должны быть пронумерованы. Блоки нумеруются слева направо. Номер работы показывается в правом нижнем углу. Используя номера блоков и оценивая влияние, которое один блок оказывает на другой, аналитик может организовать модель по принципу функционального доминирования. Это позволяет согласовать иерархический порядок функций в модели с уровнем влияния каждой функции на остальную часть системы.

Для связи работ между собой используются внутренние стрелки, то есть стрелки, которые начинаются у одной и кончаются у другой работы. В IDEF0

есть пять типов взаимосвязей между блоками: выход-вход, выход-управление, обратная связь по управлению, обратная связь по входу, выход-механизм. Примеры связей приведены на рисунке 2.



а – выход-вход; *б* – выход-управление; *в* – обратная связь по входу; *г* – обратная связь по управлению; *д* – выход-механизм

Рисунок 2 – Взаимосвязи между блоками

Связь выход-вход возникает, когда стрелка выхода вышестоящей работы направляется на вход нижестоящей.

Связь выход-управление появляется, когда выход вышестоящей работы направляется на управление нижестоящей. Связь по управлению показывает доминирование вышестоящей работы. Данные или объекты выхода вышестоящей работы не меняются в нижестоящей.

Обратная связь по входу используется, когда выход нижестоящей работы направляется на вход вышестоящей. Такая связь используется для описания циклов.

Обратная связь по управлению применяется, когда выход нижестоящей работы направляется на управление вышестоящей. Такая связь свидетельствует об эффективности бизнес-процессов.

Связь выход-механизм возникает, когда выход одной работы направляется на механизм другой. Эта взаимосвязь используется реже остальных и показывает, что одна работа подготавливает ресурсы, необходимые для проведения другой работы.

Объекты, порождённые одной работой, могут использоваться сразу в нескольких других работах. Стрелки, порождённые в разных работах, могут представлять собой однородные объекты, применяемые в одном месте. Для моделирования таких ситуаций используются разветвляющиеся и сливающиеся

стрелки. Дуги могут разветвляться и соединяться различными способами. Вся дуга или её часть может выходить из одного или нескольких блоков и заканчиваться в одном или нескольких блоках. Смысл разветвляющихся и сливающихся стрелок передаётся именованием каждой ветви стрелок. Существуют правила именования таких стрелок. Если стрелка именована до разветвления, а после разветвления ни одна из ветвей не именована, то подразумевается, что каждая ветвь моделирует те же данные или объекты, что и ветвь до разветвления.

Если стрелка именована до разветвления, а после разветвления какая-либо из ветвей тоже именована, то подразумевается, что эти ветви соответствуют именованию. Если при этом какая-либо ветвь после разветвления осталась неименованной, то подразумевается, что она моделирует те же данные или объекты, что и ветвь до разветвления. Недопустима ситуация, когда стрелка до разветвления не именована, а после разветвления не именована какая-либо из ветвей.

Правила именования сливающихся стрелок аналогичны.

Для изображения малозначимых объектов, которые не требуется изображать на всех уровнях диаграмм, может быть применён туннель. Выходом является туннелирование стрелки на самом нижнем уровне. Такое туннелирование называется «не-в-родительской-диаграмме».

Другим примером туннеля может быть ситуация, когда стрелка механизма мигрирует на нижний уровень, а механизм используется одинаково во всех работах. В этом случае стрелка на нижнем уровне удаляется, на родительской диаграмме она туннелируется, а в комментарии к стрелке можно указать, что механизм будет использоваться во всех работах дочерней диаграммы. Такое туннелирование называется «не-в-дочерней-работе».

Туннельная стрелка изображается с круглыми скобками на конце.

Пример диаграммы декомпозиции с ветвлением стрелок и туннелем представлен на рисунке 3.

Все работы модели нумеруются. Номер состоит из префикса и числа. Можно использовать любой префикс, но обычно – префикс А. Контекстная работа имеет номер А0. Работы декомпозиции А0 имеют номера А1, А2 и т. д. Работы декомпозиции нижнего уровня имеют номер родительской работы и очередной порядковый номер, например, работы декомпозиции А3 будут иметь номера А31, А32, А33 и т. д. Работы образуют иерархию, где каждая работа может иметь одну родительскую и несколько дочерних работ, образуя дерево.

Работа в IDEF0 предполагает последовательное улучшение описания системы. Но процесс улучшения в некоторый момент должен быть прекращён. Декомпозиция прекращается, когда диаграммы, образующие нижний уровень модели, достаточно детализированы для достижения цели.

Списки объектов системы, создаваемые в ходе моделирования, принято называть «списками данных». Термин «данное» здесь употребляется как синоним слова «объект». Списки данных помогают выполнить более глубокий анализ системы и при этом не концентрироваться на функциях системы. После

составления списка данных на его основе составляется список функций. Для этого представляются функции системы, использующие тот или иной вид данных. Несколько различных видов данных могут использоваться одной функцией. На базе полученных списков данных и функций формулируются основные требования к функциональности информационно-аналитической системы и типам данных, с которыми система будет работать.

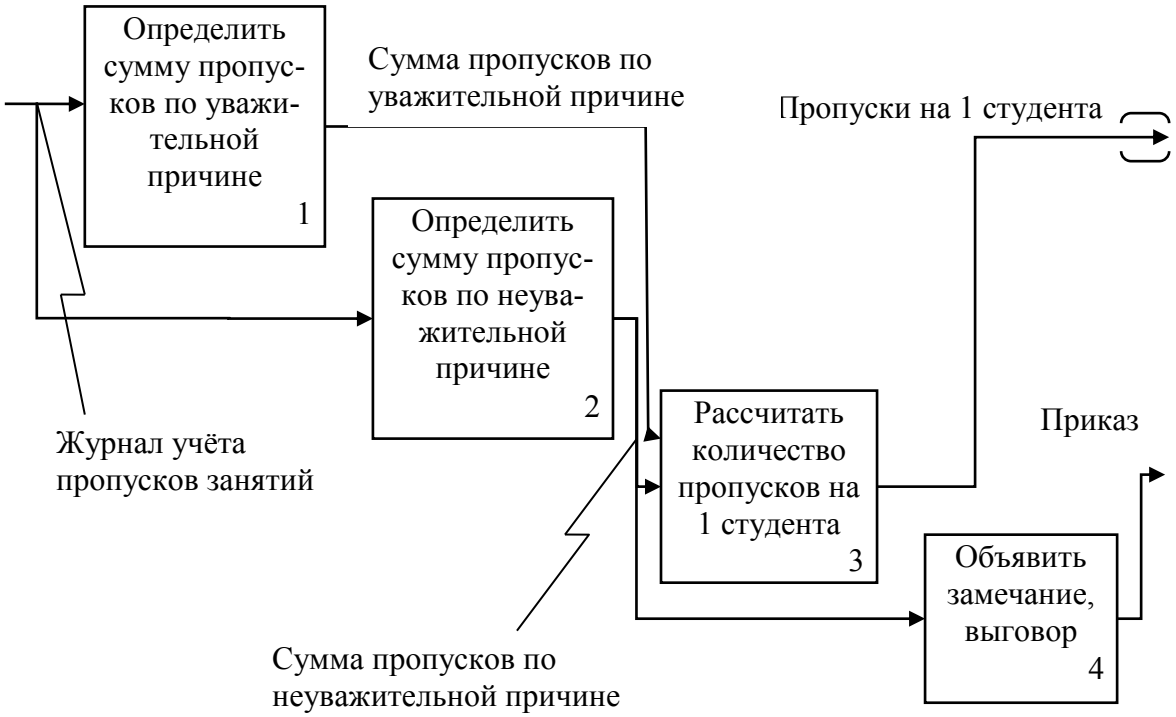


Рисунок 3 – Пример диаграммы декомпозиции

Обычно сначала строится модель существующей организации работы – AS-IS (как есть). Анализ этой модели позволяет понять, где находятся слабые места, в чём будут состоять преимущества новых бизнес-процессов и насколько глубоким изменениям подвергнется существующая структура организации бизнеса. Детализация бизнес-процессов позволяет выявить недостатки организации даже там, где функциональность кажется очевидной. Найденные в модели AS-IS недостатки можно исправить при создании модели TO-BE (как будет) – модели новой организации бизнес-процессов.

На базе анализа построенных моделей бизнес-процессов определяются область применения проектируемой информационно-аналитической системы (её назначение, границы, взаимодействие с внешней средой), основные требования к информационно-аналитической системе в части выполняемых ею функций (главные сценарии поведения системы), виды представления входной и выходной информации, требования к техническим средствам.

4.2 Построение диаграммы прецедентов

В процессе моделирования человек упрощает реальность, чтобы лучше понять проектируемую систему. С помощью UML можно строить модели из базовых блоков, таких как классы, интерфейсы, кооперации, компоненты и узлы, зависимости, обобщения и ассоциации.

Диаграмма – это графическое представление совокупности элементов, чаще всего изображаемое в виде связного графа, состоящего из вершин (сущностей) и рёбер (отношений). С помощью диаграмм можно визуализировать систему с различных точек зрения. Поскольку сложное целое нельзя понять, глядя на него лишь с одной стороны, в UML определено много разных диаграмм, которые позволяют сосредоточиться на различных аспектах моделируемой системы.

Хорошие диаграммы облегчают понимание модели. Продуманный выбор диаграмм при моделировании системы позволяет задавать правильные вопросы, помогает грамотной постановке задачи и проясняет последствия принимаемых решений.

Существует пять взаимно дополняющих друг друга видов, особенно важных для визуализации, специфицирования, конструирования и документирования программной архитектуры: это представления с точки зрения прецедентов, проектирования, процессов, реализации и развёртывания. Каждое из них предполагает моделирование статических и динамических сущностей. В своей совокупности эти виды позволяют передать наиболее важные решения, касающиеся системы в целом, а по отдельности каждый из них акцентирует внимание на одном её аспекте, рассмотрение которого таким образом упрощается.

Чтобы изобразить систему с какой-либо точки зрения средствами UML, для организации представляющих интерес элементов используются диаграммы. Правильный выбор совокупности диаграмм позволит задать нужные вопросы, касающиеся системы, и выявить риск, который необходимо учесть. Например, если моделируется простое приложение, выполняемое на одном компьютере, могут потребоваться только перечисленные далее диаграммы:

- вид с точки зрения вариантов использования – диаграммы прецедентов;
- вид с точки зрения проектирования – диаграммы классов (для структурного моделирования) и диаграммы деятельности или взаимодействия (для моделирования поведения);
- вид с точки зрения процессов – не требуется;
- вид с точки зрения реализации – не требуется;
- вид с точки зрения развертывания – также не требуется.

Системы не существуют в изоляции. Как правило, они взаимодействуют с актёрами – людьми или программами – которые используют систему в своих целях, причем каждый актёр ожидает, что она будет вести себя определённым, вполне предсказуемым образом. Прецедент (UseCase) специфицирует поведение системы или её части и представляет собой описание множества последовательностей действий, выполняемых системой для того, чтобы актёр

мог получить определённый результат.

С помощью прецедентов можно описать поведение разрабатываемой системы, не определяя её реализацию. Таким образом, они позволяют достичь взаимопонимания между разработчиками, экспертами и конечными пользователями продукта. Кроме того, прецеденты помогают проверить архитектуру системы в процессе её разработки.

На диаграмме прецедентов представлены прецеденты и актёры, а также отношения между ними. Диаграммы прецедентов относятся к статическому виду системы с точки зрения прецедентов использования. Они особенно важны при организации и моделировании поведения системы.

Графически прецедент изображается в виде эллипса. Имя прецедента может состоять из любого числа букв, цифр и некоторых знаков препинания. Имя может занимать несколько строк. На практике для именования прецедентов используют короткие глагольные фразы в активной форме, обозначающие некоторое поведение и взятые из словаря моделируемой системы.

Актёров изображают в виде человеческих фигурок. Можно определить общие типы актёров (например, Клиент) и затем специализировать их (например, создать разновидность КоммерческийКлиент) с помощью отношений обобщения. Актёров можно связывать с прецедентами только отношениями ассоциации. Ассоциация между актёром и прецедентом показывает, что они общаются друг с другом, посылая или принимая сообщения.

Построение диаграммы прецедентов осуществляется следующим образом:

– выявляются актеры, взаимодействующие с тем или иным прецедентом. К числу актёров-кандидатов относятся группы, которые требуют определённого поведения для выполнения своих задач либо необходимы для выполнения функций прецедента;

– актёры организуются путём выделения общих и специализированных ролей;

– для каждого актёра рассматриваются основные пути его взаимодействия с прецедентом;

– рассматриваются альтернативные (исключительные) способы взаимодействия актёров с прецедентом;

– прецеденты организуются с применением отношений включения и расширения для выделения общего и исключительного поведения.

Пример диаграммы прецедентов представлен на рисунке 4.

Прецедент описывает, что делает система, но не определяет, каким образом она это делает. В процессе моделирования всегда важно разделять внешнее и внутреннее представления.

Можно специфицировать поведение прецедента путём описания потока событий в текстовой форме – в виде, понятном для постороннего читателя. В описание необходимо включить указание на то, как и когда прецедент начинается и заканчивается, когда он взаимодействует с актёрами и какими объектами они обмениваются. Важно обозначить также основной и

альтернативный потоки поведения системы.

Например, моделируя работу банкомата, можно описать прецедент ПроверитьПользователя.

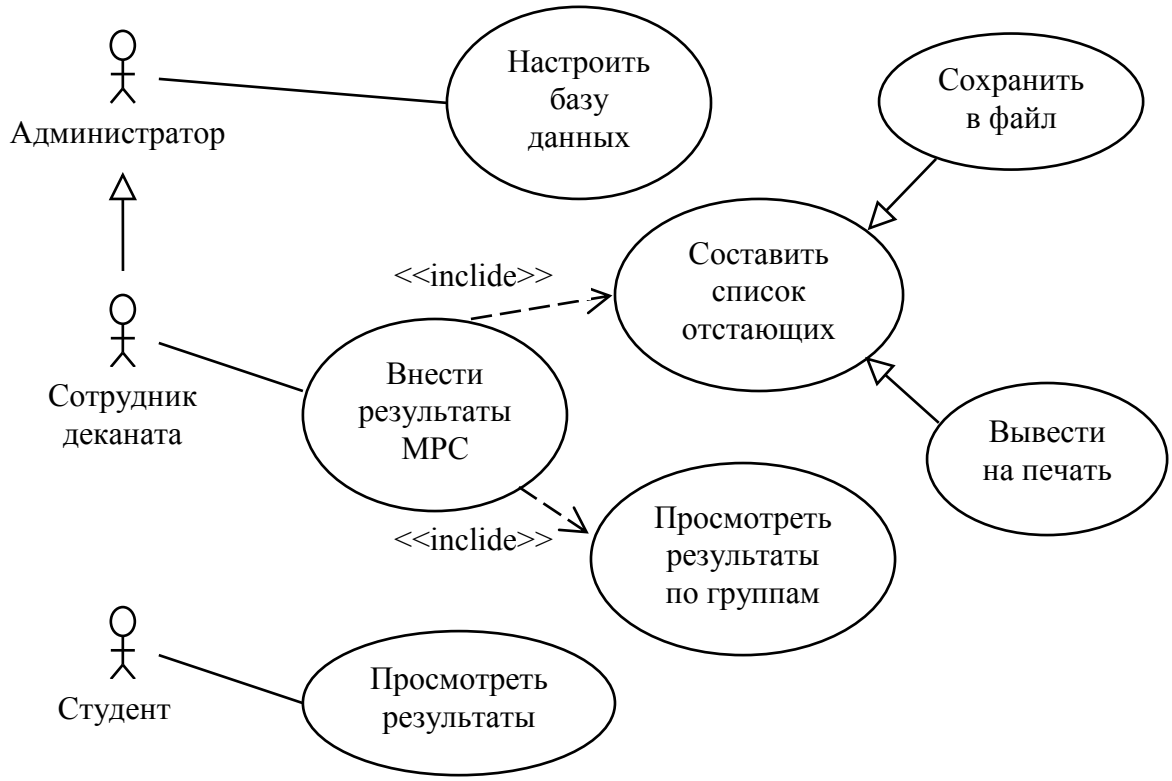


Рисунок 4 – Диаграмма прецедентов

Основной поток событий. Прецедент начинается, когда система запрашивает у клиента его персональный идентификационный номер (PIN) (E_1). Клиент может ввести его с клавиатуры (E_2). Завершается ввод нажатием клавиши Enter. После этого система проверяет введенный PIN (E_3) и, если он правильный, подтверждает ввод. На этом прецедент заканчивается.

Исключительный поток событий E_1. Клиент может прекратить транзакцию в любой момент, нажав клавишу Cancel. Это действие начинает прецедент заново. Никаких изменений на счету клиента не производится.

Исключительный поток событий E_2. Клиент может в любой момент до нажатия клавиши Enter стереть свой PIN и ввести новый.

Исключительный поток событий E_3. Если клиент ввёл неправильный PIN, прецедент запускается сначала. Если это происходит три раза подряд, система отменяет всю транзакцию и не позволяет данному клиенту снова начать работу с банкоматом в течение 60 секунд.

Поток событий в прецеденте можно описать различными способами, в том числе в виде неформализованного структурированного текста (как в примере выше), формализованного структурированного текста или с помощью псевдокода. Как правило, в начале работы потоки событий прецедента описывают в текстовой форме. По мере уточнения требований к системе будет удобнее перейти к графическому изображению потоков на диаграммах

взаимодействия. Обычно для описания основного потока прецедента используют диаграмму последовательностей, а для дополнительных – её варианты. Относительно сложная система содержит несколько десятков прецедентов, каждый из которых может разворачиваться в несколько десятков сценариев. Для любого прецедента можно выделить основные сценарии, описывающие важнейшие последовательности, и вспомогательные, описывающие альтернативные последовательности.

4.3 Построение диаграммы классов

Классы – это самые важные строительные блоки любой объектно-ориентированной системы. Они представляют собой описание совокупности объектов с общими атрибутами, операциями, отношениями и семантикой. Графически класс изображается в виде прямоугольника. У каждого класса должно быть имя, отличающее его от других классов. Имя класса – это текстовая строка, содержащая различные символы. Имя может занимать одну или несколько строк. На практике для именования класса используют одно или несколько коротких существительных, взятых из словаря моделируемой системы. Обычно каждое слово в имени класса пишется с заглавной буквы, например: Customer (Клиент), TemperatureSensor (ДатчикТемпературы).

Для класса можно указать список атрибутов. Атрибут – это именованное свойство класса, включающее описание множества значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов или не иметь их вовсе. Атрибут представляет некоторое свойство моделируемой сущности, общее для всех объектов данного класса. Например, при моделировании клиентов можно задавать фамилию, адрес, номер телефона и дату рождения. Атрибуты представляются в разделе, который расположен под именем класса.

Также для класса можно указать операции. Операция – это описание того, что позволено делать с объектом. У всех объектов класса имеется общий набор операций. Класс может содержать любое число операций или не содержать их вовсе. Например, для всех объектов класса Rectangle (Прямоугольник) из библиотеки для работы с окнами, содержащейся в пакете awt языка Java, определены операции перемещения, изменения размера и опроса значений свойств. Часто (хотя не всегда) обращение к операции объекта изменяет его состояние или его данные. Операции класса изображаются в разделе, расположенном под разделом с атрибутами класса.

Классы используются для составления словаря разрабатываемой программной системы. Это могут быть абстракции, являющиеся частью предметной области, либо классы, на которые опирается реализация. С их помощью описывают программные, аппаратные или чисто концептуальные сущности.

При построении модели программной системы оказывается, что классы редко существуют автономно. Как правило, они разными способами взаимодействуют между собой. Это значит, что в ходе моделирования

необходимо не только идентифицировать сущности, составляющие словарь системы, но и описать, как они соотносятся друг с другом.

Существуют три вида отношений, особенно часто применяемых в диаграммах классов:

– зависимости, которые описывают существующие между классами отношения использования (включая отношения уточнения, трассировки и связывания). Графически отображаются пунктирными стрелками;

– обобщения, связывающие обобщённые классы со специализированными. Графически отображаются в виде стрелки с белым окончанием;

– ассоциации, представляющие структурные отношения между объектами. Графически отображаются как прямая линия.

Каждый из этих типов отношений позволяет по-разному комбинировать классы на диаграммах.

Диаграммой классов называют диаграмму, на которой показано множество классов, интерфейсов, кооперaций и отношений между ними. Её изображают в виде множества вершин и дуг. Пример диаграммы классов представлен на рисунке 5.

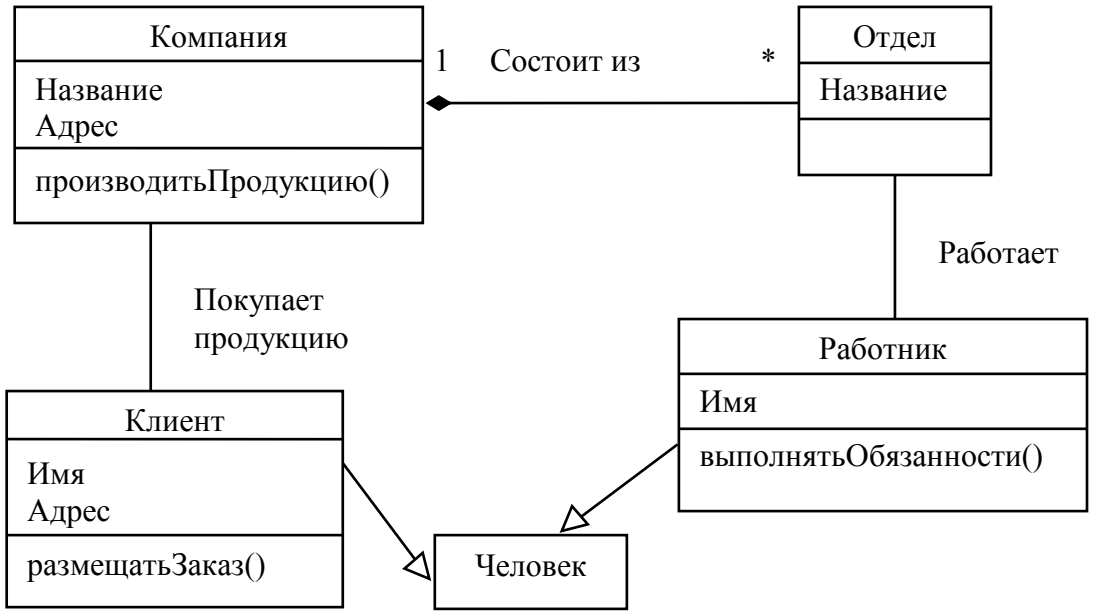


Рисунок 5 – Диаграмма классов

Диаграммы классов применяют для моделирования статического вида системы. Обычно диаграммы классов используются в следующих целях:

– для моделирования словаря системы. Оно предполагает принятие решения о том, какие абстракции являются частью системы, а какие – нет. С помощью диаграмм классов можно определить эти абстракции и их обязанности;

– для моделирования логической схемы базы данных. Логическую схему можно представить как чертёж концептуального проекта базы данных с

помощью диаграмм классов.

Многие моделируемые системы содержат устойчивые объекты, которые можно сохранять в базе данных и извлекать при необходимости. Для этого чаще всего используют реляционные, объектно-ориентированные или гибридные объектно-реляционные базы данных. UML позволяет моделировать логические схемы баз данных и сами физические базы, т. к. включает в себя как частный случай диаграммы «сущность-связь» (ER-диаграммы), которые часто используются для логического проектирования баз данных. Но если в классических ER-диаграммах внимание сосредоточено только на данных, то диаграммы классов позволяют моделировать поведение. В реальной базе данных эти логические операции трансформируются в триггеры или хранимые процедуры. Моделирование схемы производится следующим образом:

– определить классы модели, состояние которых должно сохраняться после завершения работы создавшего их приложения;

– создать содержащую эти классы диаграмму и описать их как устойчивые с помощью стандартного помеченного значения persistent;

– раскрыть структурные особенности классов. В общем случае надо детально описать их атрибуты и обратить внимание на ассоциации и их кратности;

– найти типичные структурные образцы, усложняющие проектирование физической базы данных, например циклические ассоциации, ассоциации «один к одному» и n-арные ассоциации. При необходимости создать промежуточные абстракции для упрощения логической структуры;

– рассмотреть поведение этих классов, раскрывая операции, важные для доступа к данным и поддержания их целостности;

– стараться использовать в работе инструментальные средства, позволяющие преобразовать логический проект в физический.

Содержание диаграмм классов без труда отображается на объектно-ориентированные языки программирования. Перевод диаграммы классов на программный язык производится следующим образом:

– определить правила отображения модели на один или несколько языков программирования по выбору;

– в зависимости от семантики выбранных языков придётся отказаться от использования некоторых возможностей UML;

– применять помеченные значения;

– пользоваться инструментальными средствами для отображения модели на объектно-ориентированные языки программирования.

Хорошо структурированная диаграмма классов:

– заостряет внимание только на одном аспекте статического вида системы;

– содержит лишь элементы, существенные для понимания данного аспекта;

– показывает детали, соответствующие требуемому уровню абстракции;

– не настолько лаконична, чтобы ввести в заблуждение.

Принципы построения диаграммы классов:

- давать диаграмме имя, связанное с её назначением;
- располагать элементы так, чтобы минимизировать число пересечений;
- пространственно организовывать элементы так, чтобы семантически близкие сущности располагались рядом;
- привлекать внимание к важным особенностям диаграммы, используя примечания и цвет;
- не показывать слишком много разных видов отношений; в диаграмме классов должны доминировать отношения какого-либо одного вида.

4.4 Построение диаграммы деятельности

Диаграммы деятельности используются для моделирования динамических аспектов поведения системы. Как правило, они применяются, чтобы промоделировать последовательные (а иногда и параллельные) шаги вычислительного процесса. С помощью диаграмм деятельности можно также моделировать жизнь объекта, когда он переходит из одного состояния в другое в разных точках потока управления. Диаграммы деятельности могут использоваться самостоятельно для визуализации, специфицирования, конструирования и документирования динамики совокупности объектов, но они пригодны также и для моделирования потока управления при выполнении некоторой операции.

Диаграмма деятельности – это своеобразная блок-схема, которая описывает последовательность выполнения операций во времени.

Деятельность – это некоторый относительно продолжительный этап выполнения в автомате. В конечном итоге деятельность сводится к некоторому действию, которое составлено из атомарных вычислений, приводящих к изменению состояния системы или возврату значения. Действие может заключаться в вызове другой операции, посылке сигнала, создании или уничтожении объекта либо в простом вычислении – скажем, значения выражения. Изображается действие в виде прямоугольника со скруглёнными углами.

Когда действие или деятельность в некотором состоянии завершается, поток управления сразу переходит в следующее состояние действия или деятельности. Для описания этого потока используются переходы, показывающие путь из одного состояния действия или деятельности в другое. В UML переход представляется простой линией со стрелкой. Поток управления должен где-то начинаться и заканчиваться (разумеется, если это не бесконечный поток, у которого есть начало, но нет конца). На диаграмме деятельности можно задать как начальное состояние (закрашенный кружок), так и конечное (закрашенный кружок внутри окружности).

Простые последовательные переходы встречаются наиболее часто, но их одних недостаточно для моделирования потока управления. Как и в блок-схеме, можно включить в модель ветвление, которое описывает различные пути выполнения в зависимости от значения некоторого булевского выражения. Графически точка ветвления представляется ромбом. В точку ветвления может

входить ровно один переход, а выходить – два или более. Для каждого исходящего перехода задается булевское выражение, которое вычисляется только один раз при входе в точку ветвления. Ни для каких двух исходящих переходов эти сторожевые условия не должны одновременно принимать значение «истина», иначе поток управления окажется неоднозначным. Но эти условия должны покрывать все возможные варианты, иначе поток остановится.

Реализовать итерацию на диаграмме деятельности можно, если ввести два состояния действия – в первом устанавливается значение счетчика, во втором оно увеличивается – и точку ветвления, вычисление в которой показывает, следует ли прекратить итерации.

Простые и ветвящиеся последовательные переходы в диаграммах деятельности используются чаще всего. Однако можно встретить и параллельные потоки, и это особенно характерно для моделирования бизнес-процессов. В UML для обозначения разделения и слияния таких параллельных потоков выполнения используется синхронизационная черта, которая рисуется в виде жирной вертикальной или горизонтальной линии. Каждый из параллельно выполняющихся потоков управления существует в контексте независимого активного объекта, который, как правило, моделируется либо процессом, либо вычислительной нитью.

Должен поддерживаться баланс между точками разделения и слияния. Это означает, что число потоков, исходящих из точки разделения, должно быть равно числу потоков, приходящих в соответствующую ей точку слияния. Деятельности, выполняемые в параллельных потоках, могут обмениваться друг с другом информацией, посылая сигналы. Такой способ организации взаимодействия последовательных процессов называется сопрограммами.

При моделировании течения бизнес-процессов иногда бывает полезно разбить состояния деятельности на диаграммах деятельности на группы, каждая из которых представляет отдел компании, отвечающий за ту или иную работу. В UML такие группы называются дорожками, поскольку визуально каждая группа отделяется от соседних вертикальной чертой, как плавательные дорожки в бассейне.

Каждой присутствующей на диаграмме дорожке присваивается уникальное имя. Никакой глубокой семантики дорожка не несет, разве что может отражать некоторую сущность реального мира. Каждая дорожка представляет сферу ответственности за часть всей работы, изображённой на диаграмме, и в конечном счёте может быть реализована одним или несколькими классами. На диаграмме деятельности, разбитой на дорожки, каждая деятельность принадлежит ровно одной дорожке, но переходы могут пересекать границы дорожек.

Имеется некоторая связь между дорожками и параллельными потоками выполнения. Концептуально деятельность внутри каждой дорожки обычно – но не всегда – рассматривается отдельно от деятельности в соседних дорожках. Это разумно, поскольку в реальном мире подразделения организации, представленные дорожками, как правило, независимы и функционируют параллельно. Пример диаграммы деятельности представлен на рисунке 6.

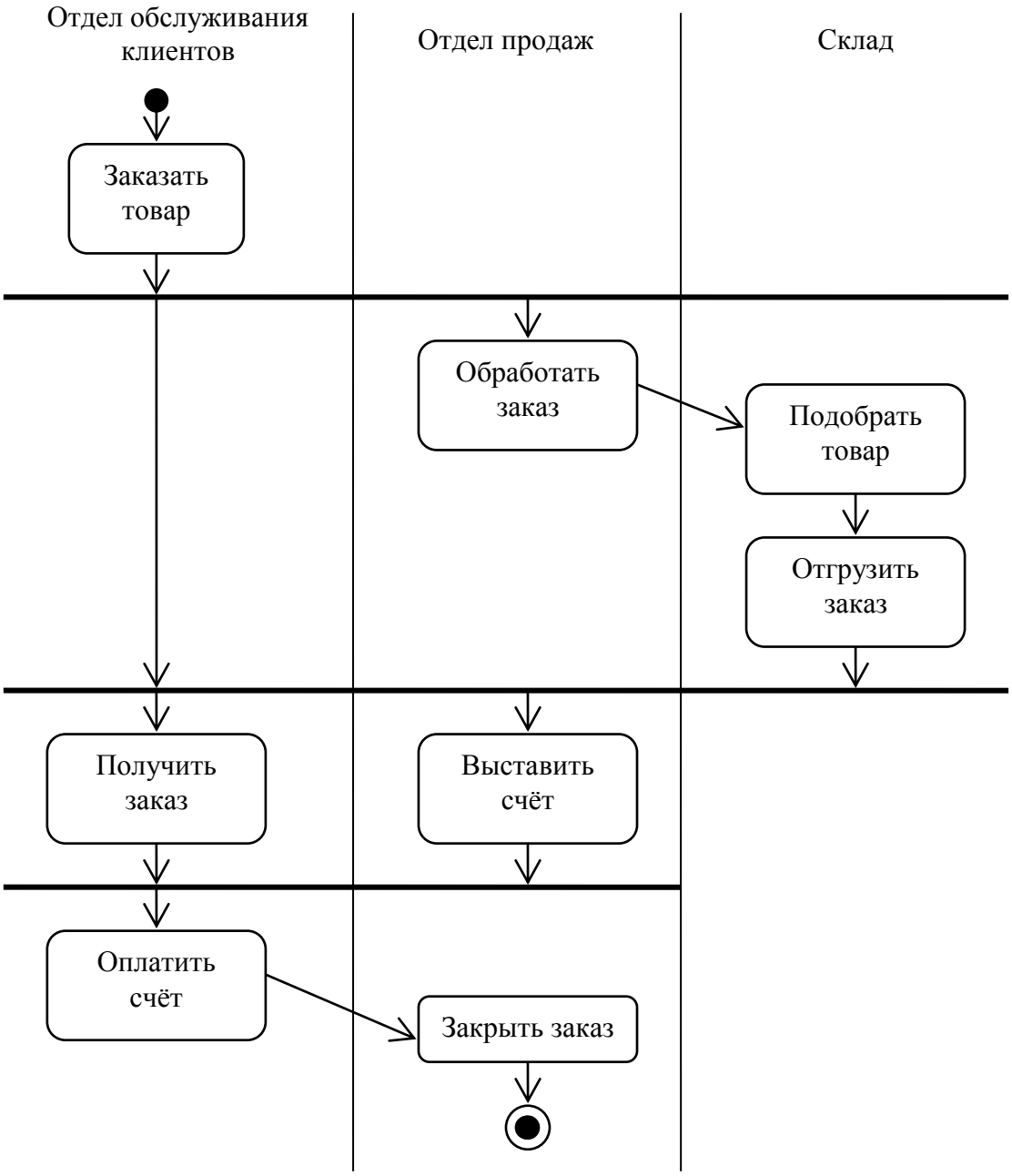


Рисунок 6 – Диаграмма деятельности

Использовать диаграммы деятельности для моделирования некоторого динамического аспекта системы можно в контексте любого элемента модели. Но чаще всего они рассматриваются в контексте системы в целом, подсистемы, операции или класса. Можно присоединять диаграммы деятельности к прецедентам и кооперациям. При моделировании динамических аспектов системы диаграммы деятельности применяются в основном двумя способами:

– для моделирования рабочего процесса. Здесь внимание фокусируется на деятельности с точки зрения актёров, которые сотрудничают с системой. Рабочие процессы часто оказываются с внешней, обращённой к пользователю стороны системы, и используются для описания бизнес-процессов разрабатываемой системы. При этом особенно важное значение имеет моделирование траекторий объектов;

– для моделирования операции. В этом случае диаграммы деятельности используются как блок-схемы для моделирования деталей вычислений. Для такого применения особенно важно моделирование точек ветвления, разделения и слияния. При этом контекст диаграммы деятельности включает параметры операции и её локальные объекты.

Программная система не существуют изолированно; всегда имеется некоторый контекст, в рамках которого она функционирует, причем он включает актёров, взаимодействующих с системой. Как правило, программное обеспечение работает в контексте бизнес-процессов более высокого уровня (рабочих процессов). Моделировать эти бизнес-процессы можно с помощью диаграмм деятельности. Для того чтобы построить модель рабочего процесса, следует:

1) выделить какой-либо участок рабочего процесса. В сложных системах невозможно отразить все последовательности на одной диаграмме;

2) выбрать бизнес-объекты, на которые возложена ответственность высокого уровня за части всего рабочего процесса. Это могут быть реальные сущности, вошедшие в системный словарь, или более абстрактные объекты. Следует создать отдельную дорожку для каждого бизнес-объекта;

3) определить предусловия для начального состояния рабочего процесса и постусловия для его конечного состояния, чтобы задать границы процесса;

4) начиная с исходного состояния описать деятельности и действия, выполняемые в различные моменты времени, а затем отразить их на диаграмме деятельности в виде состояний деятельности или действий;

5) сложные действия или множества действий, встречающиеся многократно, следует свернуть в состояния деятельности и для каждого из таких состояний составить отдельную диаграмму деятельности;

6) изобразить переходы, соединяющие состояния деятельностей и действий. Сначала нужно сосредоточиться на последовательных потоках, затем перейти к ветвлениям и далее рассмотреть разделения и слияния;

7) если в рабочий процесс вовлечены важные объекты, изобразить их на диаграмме деятельности. Иногда следует показать изменение значений и состояний таких объектов, чтобы прояснить суть траектории каждого.

Диаграмму деятельности можно присоединить к любому элементу модели для описания поведения этого элемента. Чаще всего диаграммы деятельности присоединяются к операциям. В этом случае они становятся блок-схемой выполняемых действий. Главное преимущество диаграммы деятельности заключается в том, что все её элементы семантически связаны с лежащей в её основе богатой моделью. Моделирование операции состоит из следующих шагов:

1) выявить абстракции, относящиеся к операции. Сюда относятся параметры операции (включая тип возвращаемого значения, если таковое имеется), атрибуты объемлющего класса и некоторых соседних классов;

2) определить предусловия в начальном состоянии и постусловия в конечном состоянии операции;

3) начиная с исходного состояния операции описать деятельности и

действия, протекающие во времени, и изобразить их на диаграмме деятельности в виде состояний деятельности или действий;

4) при необходимости использовать точки ветвления для описания условных переходов и итераций;

5) лишь в том случае, если владельцем операции является активный класс, использовать точки разделения и слияния для описания параллельных потоков выполнения, если есть необходимость.

Использование диаграмм деятельности в качестве блок-схем превращает UML в язык визуального программирования. Можно нарисовать блок-схему для каждой операции, но более естественно кодировать тело операции на некотором языке программирования. Использование диаграмм деятельности для моделирования операции становится разумным, когда эта операции сложна, так что разобраться в ней, глядя только на код, достаточно трудно.

При создании диаграмм деятельности следует помнить, что они моделируют срез некоторых динамических аспектов поведения системы. С помощью одной диаграммы деятельности невозможно охватить все динамические аспекты системы. Вместо этого следует использовать разные диаграммы деятельности для моделирования динамики рабочих процессов или отдельных операций.

Хорошо структурированная диаграмма деятельности:

- сконцентрирована на описании одного аспекта динамики системы;
- содержит только элементы, существенные для понимания этого аспекта;
- представляет лишь детали, соответствующие своему уровню абстракции;
- не настолько краткая, чтобы читатель упустил из виду важные аспекты.

Принципы построения диаграммы деятельности:

- давать диаграмме имя, соответствующее её назначению;
- начинать с моделирования главного потока. Ветвления, параллельность и траектории объектов являются второстепенными деталями, которые можно изобразить на отдельной диаграмме;
- располагать элементы так, чтобы число пересечений было минимальным;
- использовать примечания и цвет, чтобы привлечь внимание к важным особенностям диаграммы.

Список литературы

1 Информационные системы и технологии в экономике и управлении [Электронный ресурс] / Под ред. В. В. Трофимова. – Москва: КноРус, 2011. – CD.

2 **Орлов, С. А.** Технологии разработки программного обеспечения: разработка сложных программных систем: учебное пособие / С. А. Орлов. – Санкт-Петербург: Питер, 2003. – 480 с.: ил.

3 **Кантор Марри.** Управление программными проектами: практическое руководство по разработке успешного программного продукта / Марри Кантор. – Москва: Вильямс, 2002. – 176 с.

4 **Саак, А.** Информационные технологии управления : учебник для вузов / А. Саак, Е. В. Пахомов, В. Н. Тюшняков. – 2-е изд. – Санкт-Петербург: Питер, 2010. – 320 с. + CD.

5 **Гордеев, А. В.** Системное программное обеспечение: учебник / А. В. Гордеев. – Санкт-Петербург: Питер, 2002. – 736 с.

6 **Кинг, Д.** Создание эффективного программного обеспечения / Д. Кинг. – Москва: Мир, 1991. – 288 с.: ил.