

ВОЕННО-КОСМИЧЕСКАЯ АКАДЕМИЯ
имени А.Ф. Можайского

В.А. Мыльников, В.А. Лохвицкий, А.С. Марковский

**Технология разработки
программного обеспечения**

Практикум

*Учебное издание «Технология разработки программного обеспечения»
утверждено в качестве практикума и рекомендовано кафедрой
математического и программного обеспечения Военно-космической
академии имени А.Ф.Можайского для обучающихся по основным
профессиональным образовательным программам высшего образования –
программам специалитета по направлению подготовки «Применение
и эксплуатация автоматизированных систем специального
назначения», протокол от 05 сентября 2016 г. № 2*



Санкт-Петербург
2016

Составители:
В.А. Мыльников, В.А. Лохвицкий, А.С. Марковский

Технология разработки программного обеспечения: практикум / сост.: В.А. Мыльников, В.А. Лохвицкий, А.С. Марковский; под общ. ред. В.А. Мыльникова. – СПб.: ВКА имени А.Ф. Можайского, 2016. – 150 с.

Практикум по дисциплине *«Технология разработки программного обеспечения»*, входящей в цикл специальных дисциплин, предназначен для курсантов, обучающихся по специальности «Применение и эксплуатация автоматизированных систем специального назначения», и содержит задания, связанные с проектированием и разработкой программного обеспечения и автоматизированных информационных систем.

Практикум соответствует государственному образовательному стандарту высшего образования.

Практикум подготовили: кандидат технических наук, старший научный сотрудник А.С. Марковский (практические занятия № 1–3), кандидат технических наук, доцент В.А. Мыльников (практические занятия № 4–18), кандидат технических наук, В.А. Лохвицкий (практические занятия № 19–20).

© ВКА имени А.Ф. Можайского, 2016

Подписано к печ. 07.10.2016
Гарнитура Times New Roman
Уч.-печ. л. 19

Формат печатного листа 445×300/8
Уч.-изд. л. 9,25
Заказ 3301
Бесплатно

Типография ВКА имени А.Ф. Можайского

СОДЕРЖАНИЕ

Общие методические указания курсантам (слушателям) по подготовке к практическим занятиям	5
Практическое занятие №1 Тема: «Нормативно-методическая база разработки и эксплуатации систем и программных средств»	6
Практическое занятие №2 Тема: «Разработка описания и анализ информационной системы».....	8
Практическое занятие №3 Тема: «Разработка требований к информационной системе»	16
Практическое занятие №4 Тема: «Методология функционального моделирования IDEF0. Построение контекстной диаграммы и декомпозиции»	26
Практическое занятие №5 Тема: «Туннелирование стрелок. Расщепление и слияние модели при групповой работе»	45
Практическое занятие №6 Тема: «Методология описания процессов IDEF3»	57
Практическое занятие №7 Тема: «Стоимостный анализ и свойства, определяемые пользователем»	64
Практическое занятие №8 Тема: «Разработка модели потоков данных DFD информационной системы»	72
Практическое занятие №9 Тема: «Разработка структуры данных методом ER-диаграмм».....	79
Практическое занятие №10 Тема: «Прямое и обратное проектирование базы данных».....	87
Практическое занятие №11 Тема: «Диаграмма классов (Class Diagram) и диаграмма объектов (Object Diagram)».....	98

Практическое занятие №12	
Тема: «Диаграммы вариантов использования (Use-Case diagram)».....	105
Практическое занятие №13	
Тема: «Диаграммы состояний (StateChart diagram)».....	109
Практическое занятие №14	
Тема: «Диаграммы последовательностей (Sequence Diagram)».....	113
Практическое занятие №15	
Тема: «Диаграммы коммуникации (Collaboration Diagram)».....	119
Практическое занятие №16	
Тема: «Диаграммы видов деятельности (Activity Diagram)».....	124
Практическое занятие №17	
Тема: «Диаграмма компонентов (Component Diagram)».....	128
Практическое занятие №18	
Тема: «Диаграмма развертывания (Deployment Diagram) и диаграмма пакетов (Packages Diagram)»	131
Практическое занятие №19	
Тема: «Методология тестирования информационной системы».....	135
Практическое занятие №20	
Тема: «Совместная разработка информационной системы».....	143
Список литературы.....	150

ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ КУРСАНТАМ (СЛУШАТЕЛЯМ) ПО ПОДГОТОВКЕ К ПРАКТИЧЕСКИМ ЗАНЯТИЯМ

Практические занятия по дисциплине «Технология разработки программного обеспечения» проводятся в классе ПЭВМ. Индивидуальные задания выполняются лично каждым курсантом (слушателем).

Перед выполнением задания обучающийся изучает материал, приведенный в разделе «Теоретические основы занятия», после чего приступает к изучению материалов раздела «Экспериментальная часть занятия», в ходе которого необходимо разобрать приведенные примеры и выполнить задания раздела. На следующем этапе работы обучающийся выполняет индивидуальное задание.

Результаты работы оформляются в виде отчета. Содержание отчета приведено в руководстве по соответствующему практическому занятию.

О готовности к защите работы курсант докладывает преподавателю.

Практическое занятие № 1

ТЕМА: «НОРМАТИВНО-МЕТОДИЧЕСКАЯ БАЗА РАЗРАБОТКИ И ЭКСПЛУАТАЦИИ СИСТЕМ И ПРОГРАММНЫХ СРЕДСТВ»

Цель занятия: ознакомиться с нормативно-методической базой и классификацией систем и программных средств.

Задачи:

1. Изучить основные стандарты, определяющие этапы жизненного цикла систем и программных средств.
2. Изучить основные стандарты, регламентирующие вопросы контроля качества систем и программных средств.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Ознакомление с нормативными документами.

Теоретические основы занятия

Коллективная разработка в отличие от индивидуальной требует четкого планирования работ и их распределения во время создания программной системы. Один из способов организации работ состоит в разбиении процесса разработки на отдельные последовательные стадии, после полного прохождения которых получается конечный продукт или его часть. Такие стадии называют жизненным циклом разработки программной системы. Как правило, жизненный цикл начинается с формирования общего представления о разрабатываемой системе и их формализации в виде требований верхнего уровня. Завершается жизненный цикл разработки вводом системы в эксплуатацию. Однако нужно понимать, что разработка – только один из процессов, связанных с программной системой, которая также имеет свой жизненный цикл (ЖЦ). В отличие от жизненного цикла разработки системы жизненный цикл самой системы заканчивается выводом ее из эксплуатации и прекращением ее использования.

Жизненный цикл программного обеспечения – это совокупность итерационных процедур, связанных с последовательным изменением состояния программного обеспечения от формирования исходных требований к нему до окончания его эксплуатации конечным пользователем.

Процесс создания и конфигурирования для различных систем включает в себя один набор этапов, благодаря чему можно говорить о моделях ЖЦ.

Модель жизненного цикла системы – это комбинация последовательности этапов жизненного цикла и переходов между ними, необходимых для гарантированного достижения поставленной для реализации проекта цели.

В настоящее время фонд документов по стандартизации оборонной продукции (ДСОП), созданный совместной работой Ростехрегулирования, Минобороны России и отраслей промышленности, составляет более 45 тыс. документов. Фонд документов в области общетехнических и организационно-методических систем (комплексов) стандартов на оборонную продукцию (ДСОП) составляет более 1,3 тыс. документов. Перечень наиболее важных из них можно разделить на две большие группы, представленные в табл. 1.1 и 1.2.

Таблица 1.1

**Стандарты, определяющие этапы жизненного цикла систем
и программных средств**

ГОСТ РВ 0015–002–2012	Система разработки и постановки на производство военной техники. Система менеджмента качества. Общие положения
ГОСТ РВ 15.203-2001	СРПП. Порядок выполнения ОКР
ГОСТ РВ 15.201-2003	Тактико-техническое задание на ОКР
ГОСТ 34.601-90	Автоматизированные системы. Стадии создания
ГОСТ Р 56135-2014	Управление жизненным циклом продукции военного назначения
ГОСТ Р 51189-98	Порядок разработки ПССВ
ГОСТ Р ИСО/МЭК 12207-2010	СиПИ. Процессы ЖЦ программных систем
ГОСТ Р ИСО/МЭК 15288-2005	СиПИ. Процессы ЖЦ систем

Таблица 1.2

Стандарты, регламентирующие вопросы контроля качества систем и ПС

ГОСТ 28195-89	Оценка качества программных средств. Общие положения
ГОСТ Р ИСО/МЭК 9126-93	Характеристики качества и руководства по их применению
ГОСТ РВ 51718-2001	Программное изделие. Общие технические требования
ГОСТ РВ 51719-2001	Программное изделие. Испытания программной продукции. Общие технические требования
ГОСТ Р ИСО/МЭК 25021-2014	СиПИ. Требования и оценка качества систем и программного обеспечения (SQuaRE). Элементы показателя качества
ГОСТ Р ИСО/МЭК 25041-2012	СиПИ. Требования и оценка качества систем и программного обеспечения (SQuaRE). Руководство по оценке для разработчиков, приобретателей и независимых оценщиков

Контрольные вопросы

1. Дайте определение жизненного цикла программного обеспечения.
2. Дайте определение модели жизненного цикла системы.
3. На какие группы можно разделить нормативные документы?

Практическое занятие № 2

ТЕМА: «РАЗРАБОТКА ОПИСАНИЯ И АНАЛИЗ ИНФОРМАЦИОННОЙ СИСТЕМЫ»

Цель занятия: описать и проанализировать информационную систему, распределить роли в группе разработчиков.

Задачи:

1. Составить подробное описание системы.
2. Провести анализ осуществимости проекта.
3. Распределить роли в группе между участниками команды.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Ознакомление с нормативными документами ГОСТ РВ 51718-2001, ГОСТ 51189-98, ГОСТ Р 51904 – 2002.
3. Выполнение индивидуальных заданий в соответствии с вариантом.

Теоретические основы занятия

Общие сведения

На этапе анализа информационной системы проводится анализ предметной области, для которой разрабатывается программное обеспечение (ПО).

Необходимо понимать разницу между профессиональной разработкой ПО и любительским программированием. Необходимость управления программными проектами вытекает из того факта, что процесс создания профессионального ПО всегда является субъектом бюджетной политики организации, где оно разрабатывается, и имеет временные ограничения.

Одной из важнейших задач разработки ПО является качественное проведение этапов анализа и проектирования, обеспечивающих создание модели функционирования программной системы, которая может заложить твердый фундамент для дальнейшей ее программной реализации.

Невозможно описать и стандартизировать все работы, выполняемые в проекте по созданию ПО. Эти работы весьма существенно зависят от организации, где выполняется разработка ПО, и от типа создаваемого программного продукта, но всегда можно выделить следующие:

1. Написание предложений по созданию ПО. Оно может состоять из написания предложений по реализации этого проекта. Предложения должны содержать описание целей проектов и способов их достижения. Они также обычно включают в себя оценки финансовых и временных затрат на выполнение проекта. При необходимости здесь могут приводиться обоснования для передачи проекта на выполнение сторонней организации или команде разработчиков.

2. Планирование и составление графика работ по созданию ПО. При этом определяются процессы, этапы и полученные на каждом из них результаты, которые должны привести к выполнению проекта. Реализация этого графика приведет к достижению целей проекта.

3. Оценивание стоимости проекта. Оно напрямую связано с его планированием, поскольку здесь оцениваются ресурсы, требующиеся для выполнения плана.

4. Контроль за ходом выполнения работ, или мониторинг проекта. Это – непрерывный процесс, продолжающийся в течение всего срока реализации проекта. Руководитель должен постоянно отслеживать ход реализации проекта и сравнивать фактические и плановые показатели выполнения работ с их стоимостью. В течение реализации проекта обычно происходит несколько формальных контрольных проверок хода выполнения работ по созданию ПО.

5. Написание отчетов и представлений.

В рамках изучаемой дисциплины роли в группе по разработке ПО распределены следующим образом:

- *руководитель* осуществляет общее руководство проектом, написание документации, общение с заказчиком ПО;
- *системный аналитик* разрабатывает требования (составляет техническое задание, проект программного обеспечения);
- *тестер* составляет план тестирования и аттестации готового ПО, составляет сценарий тестирования, базовый пример, проводит мероприятия по плану тестирования;
- *разработчик* моделирует компоненты программного обеспечения, кодирование.

Планирование проекта разработки программного обеспечения

Эффективное управление программным проектом напрямую зависит от правильного планирования работ, необходимых для его выполнения. План, разработанный на начальном этапе проекта, рассматривается всеми его участниками как руководящий документ, выполнение которого должно привести к успешному завершению проекта. Этот первоначальный план должен максимально подробно описывать все этапы реализации проекта.

Процесс планирования начинается, исходя из описания системы, с определения проектных ограничений (временных ограничений, возможности наличного персонала, бюджетных ограничений и т.д.). Эти ограничения должны определяться параллельно с оцениванием проектных параметров, таких, как структура и размер проекта, а также распределением функций среди исполнителей. Затем определяются этапы разработки и то, какие результаты (документация, прототипы, подсистемы или версии программного продукта) должны быть получены по окончании этих этапов. Далее начинается

циклическая часть планирования. Сначала разрабатывается график работ по выполнению проекта или дается разрешение на продолжение использования ранее созданного графика. После этого проводится контроль выполнения работ и отмечаются расхождения между реальным и плановым ходом работ.

Далее, по мере поступления новой информации о ходе выполнения проекта, возможен пересмотр первоначальных оценок параметров проекта. Это, в свою очередь, может привести к изменению графика работ. Если в результате этих изменений нарушаются сроки завершения проекта, то должны быть пересмотрены и согласованы с заказчиком ПО проектные ограничения.

Конечно, большинство руководителей проектов не думают, что реализация их проектов пройдет гладко, без всяких проблем. Желательно описать возможные проблемы еще до того, как они проявят себя в ходе выполнения проекта. Поэтому лучше составлять «пессимистические» графики работ, чем «оптимистические». Но, конечно, невозможно построить план, учитывающий все, в том числе случайные, проблемы и задержки выполнения проекта, поэтому и возникает необходимость периодического пересмотра проектных ограничений и этапов создания программного продукта.

План проекта должен четко показать ресурсы, необходимые для его реализации, разделение работ на этапы и временной график выполнения этих этапов. В некоторых организациях план проекта составляется как единый документ, содержащий все виды планов, описанных выше. В других случаях план проекта описывает только технологический процесс создания ПО. В таком плане обязательно присутствуют ссылки на планы других видов, но они разрабатываются отдельно от плана проекта.

Детализация планов проектов очень разнится в зависимости от типа разрабатываемого программного продукта и организации-разработчика. Но в любом случае большинство планов содержат следующие разделы:

1. *Введение* [краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом].
2. *Организация выполнения проекта* (описание способа подбора команды разработчиков и распределение обязанностей между членами команды).
3. *Анализ рисков* (описание возможных проектных рисков, вероятности их проявления и стратегий, направленных на их уменьшение).
4. *Аппаратные и программные ресурсы, необходимые для реализации проекта* (перечень аппаратных средств и программного обеспечения, необходимого для разработки программного продукта. Если аппаратные средства требуется закупать, то приводится их стоимость совместно с графиком закупки и поставки).
5. *Разбиение работ на этапы* [процесс реализации проекта разбивается на отдельные процессы, определяются этапы выполнения проекта, приводится описание результатов («выходов») каждого этапа и контрольные отметки].

6. *График работ* [в этом графике отображаются зависимости между отдельными процессами (этапами) разработки ПО, оценка времени их выполнения и распределение членов команды разработчиков по отдельным этапам].

7. *Механизмы мониторинга и контроля за ходом выполнения проекта* (описываются предоставляемые руководителем отчеты о ходе выполнения работ, сроки их предоставления, а также механизмы мониторинга всего проекта).

План должен регулярно пересматриваться в процессе реализации проекта. Одни части плана, например график работ, изменяются часто, другие более стабильны. Для внесения изменений в план требуется специальная организация документопотока, позволяющая отслеживать эти изменения.

Общие сведения о требованиях к информационным системам

Проблемы, которые приходится решать специалистам в процессе создания программного обеспечения, очень сложны. Природа этих проблем не всегда ясна, особенно если разрабатываемая программная система инновационная. В частности, трудно четко описать те действия, которые должна выполнять система. Описание функциональных возможностей и ограничений, накладываемых на систему, называется требованиями к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений – разработкой требований.

Требования подразделяются на пользовательские и системные. Пользовательские требования – это описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на нее. Системные требования – это описание особенностей системы (архитектура системы, требования к параметрам оборудования и т.д.), необходимых для эффективной реализации требований пользователя.

Анализ осуществимости проекта

Разработка требований – это процесс, включающий мероприятия, необходимые для создания и утверждения документа, содержащего спецификацию системных требований. Для новых программных систем процесс разработки требований должен начинаться с анализа осуществимости проекта. Началом такого анализа является общее описание системы и ее назначения, а результатом анализа – отчет, в котором должна быть четкая рекомендация, продолжать или нет процесс разработки требований проектируемой системы. Другими словами, анализ осуществимости должен осветить следующие вопросы:

1. Отвечает ли система общим и бизнес-целям организации-заказчика и организации-разработчика?

2. Можно ли реализовать систему, используя существующие на данный момент технологии и не выходя за пределы заданной стоимости?

3. Можно ли объединить систему с другими системами, которые уже эксплуатируются?

Критическим является вопрос, будет ли система соответствовать целям организации. Если система не соответствует этим целям, то она не представляет никакой ценности для организации. В то же время многие организации разрабатывают системы, не соответствующие их целям, либо не совсем ясно понимая эти цели, либо под влиянием политических или общественных факторов.

Выполнение анализа осуществимости проекта включает сбор и анализ информации о будущей системе и написание соответствующего отчета. Сначала следует определить, какая именно информация необходима, чтобы ответить на поставленные выше вопросы. Например, эту информацию можно получить, ответив на следующие вопросы:

1. Что произойдет с организацией, если система не будет введена в эксплуатацию?

2. Какие текущие проблемы существуют в организации и как новая система поможет их решить?

3. Каким образом система будет способствовать целям бизнеса?

4. Требуется ли разработка системы технологии, которая до этого не использовалась в организации?

Далее необходимо определить источники информации. Это могут быть менеджеры отделов, где система будет использоваться, разработчики программного обеспечения, знакомые с типом будущей системы, технологи, конечные пользователи и т.д.

После обработки собранной информации готовится отчет по анализу осуществимости создания системы. В нем должны быть даны рекомендации относительно продолжения разработки системы. Могут быть предложены изменения бюджета и графика работ по созданию системы или предъявлены более высокие требования к системе.

Экспериментальная часть занятия

Индивидуальное задание

Цели анализа предметной области следующие:

- 1) определение границ, или контура, системы;
- 2) описание объектов автоматизации и/или формализации знаний об этих объектах;
- 3) выявление или определение потребностей заказчика ПО.

Анализ предметной области можно проводить, например, основываясь на теории системного анализа и использовать предложенные в ней методы.

Исходными данными для этапа системного анализа являются:

1) регламенты работы отделов и должностные инструкции сотрудников этих отделов;

2) анкеты опроса заинтересованных лиц;

3) записи интервью с заинтересованными лицами;

4) другие документы, имеющие отношение к исследуемому объекту.

Выходными данными, или результатом, этапа системного анализа являются:

1) перечень заинтересованных лиц;

2) список потребностей заинтересованных лиц в разрабатываемом ПО;

3) описание объектов автоматизации;

4) модель объектов автоматизации или предметной области.

По результатам проведенного анализа должны быть сформулированы потребности заказчика относительно разрабатываемого программного обеспечения. Далее необходимо провести аналогию между выявленными потребностями и структурой требований технического задания в соответствии с ГОСТ 34.602-89, ГОСТ 19.201-78 (раздел «Назначение и цели создания системы» технического задания).

Варианты заданий для индивидуальной работы

1. Абитуриент из потока стал курсантом учебной группы факультета.

2. Курсант учебной группы изучает дисциплины, сдает экзамены и зачеты.

3. Курсант учебной группы формирует график занятий по индивидуальному плану обучения.

4. Преподаватель кафедры факультета проводит занятия по дисциплинам.

5. Курсант учебной группы факультета изучает дисциплины определенной специальности и курса.

6. Курсант регистрируется в группе по специальности факультета вуза города.

7. Абитуриент одного из потоков сдает экзамены по определенным предметам и получает отметки.

8. На один из складов фирмы поступают товары от различных поставщиков.

9. На один из складов одной из фирм города поступает товар от различных поставщиков.

10. Со складов фирмы выдаются товары от различных поставщиков различным потребителям из различных городов.

11. Со складов различных фирм города выдаются товары различным потребителям из различных городов.

12. Со склада фирмы выдаются товары различных поставщиков и различных производителей различным потребителям различных городов.

13. На склад поступают товары различных производителей различных стран от поставщиков различных городов.

14. На склады различных фирм города поступает товар различных производителей от различных поставщиков.

15. Клиенты делают покупки различных товаров в магазинах торговой фирмы наличными, по карточкам и в кредит.

16. Покупатели магазина делают покупки различных товаров различных производителей и различных поставщиков.

17. Клиенты покупают товар различных производителей в магазинах торговой фирмы наличными, по карточкам и в кредит.

18. В магазине торгуют товарами различного вида, различных производителей и от различных поставщиков.

19. В магазинах торговой фирмы торгуют товарами различного вида, различных производителей и от различных поставщиков.

20. В магазинах торговой фирмы торгуют товарами различного вида наличными, по карточкам и в кредит.

21. В магазине торгуют товарами различного вида, различных производителей и от различных поставщиков наличными, по карточкам и в кредит.

22. В отделе кадров ведется учет подразделений, должностей и сотрудников; в личном деле сотрудника регистрируются приказы и изменения в положении сотрудника.

23. Проекты состоят из разного набора документов, документы учитываются и отправляются заказчику.

24. Военное обмундирование различается по классу, виду, категории военнослужащих, размеру; ведется учет поступлений и продаж.

25. Запасные части узла связи классифицируются по типу, наименованию; ведется учет наличия на складе и замены при ремонтах радиостанций.

26. Контролеры определяют качество изделий и ведут журнал учета годных, списанных и дорабатываемых изделий.

27. Контролеры регистрируют результаты тестирования партии изделий в маршрутных листах.

28. Эксперты назначают эталонные значения параметров изделия определенного типа; операторы измеряют параметры каждого изделия.

29. Изделия характеризуются эталонными параметрами, ведется журнал учета отклонений реальных параметров от эталонных.

30. ГСМ в автопарке классифицируются по типу, наименованию и характеристикам; ведется учет поступлений и списаний.

Требования к оформлению отчета

Отчет должен содержать следующие разделы:

1. Цель работы.

2. Введение [краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом].

3. Описание информационной системы (ПО) (наличие заключения о возможности реализации проекта, содержащего рекомендации относительно разработки системы, базовые предложения по объему требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению).

4. Анализ осуществимости (согласно требованиям к результатам

выполнения п.2, необходимо указать возможные проблемы и пути их решения).

5. Роли участников группы разработки ПО.
6. Программно-аппаратные средства, используемые при выполнении работы.
7. Заключение (выводы).
8. Список используемой литературы.

Контрольные вопросы

1. Какие роли выделяют для участников группы по разработке программного обеспечения?
2. Назовите основные разделы плана проекта.
3. К каким основным группам можно отнести требования к информационным системам?
4. При решении задачи анализа осуществимости проекта на какие основные вопросы следует ответить?

Практическое занятие №3

ТЕМА: «РАЗРАБОТКА ТРЕБОВАНИЙ К ИНФОРМАЦИОННОЙ СИСТЕМЕ»

Цель занятия: составить и проанализировать требования к информационной системе, разработать техническое задание на разработку программного обеспечения.

Задачи:

1. Изучить теоретический материал.
2. Сформулировать опорные точки зрения на основании метода VORD для формирования и анализа требований, разработать диаграммы идентификации и иерархии точек зрения.
3. Разработать информационную модель будущей системы, на ее основе сформировать требования пользователя и системные требования. Провести аттестацию требований, обосновать выбранный тип проверки.
4. Разработать техническое задание на создание программного обеспечения в соответствии с рекомендациями ГОСТ 34.602-89, ГОСТ 19.201-78.

Теоретические основы занятия

Общие сведения о требованиях к информационным системам

Проблемы, которые приходится решать специалистам в процессе создания программного обеспечения, очень сложны. Природа этих проблем не всегда ясна, особенно если разрабатываемая программная система инновационная. В частности, трудно четко описать те действия, которые должна выполнять система. Описание функциональных возможностей и ограничений, накладываемых на систему, называется требованиями к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений – разработкой требований.

Требования подразделяются на пользовательские и системные. Пользовательские требования – это описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на нее. Системные требования – это описание особенностей системы (архитектуры системы, требований к параметрам оборудования и т.д.), необходимых для эффективной реализации требований пользователя.

Разработка требований

Разработка требований – это процесс, включающий мероприятия, необходимые для создания и утверждения документа, содержащего спецификацию системных требований. Различают четыре основных этапа процесса разработки требований:

- 1) анализ технической осуществимости создания системы;
- 2) формирование и анализ требований;
- 3) специфицирование требований и создание соответствующей документации;
- 4) аттестацию этих требований.

На рис. 3.1 показаны взаимосвязи между перечисленными этапами и результаты, сопровождающие каждый этап процесса разработки системных требований.

Но поскольку в процессе разработки системы в силу разнообразных причин требования могут меняться, управление требованиями, т.е. процесс управления изменениями системных требований, является необходимой составной частью деятельности по их разработке.



Рис. 3.1. Процесс разработки требований

Формирование и анализ требований

Следующим этапом процесса разработки требований является формирование (определение) и анализ требований.

Обобщенная модель процесса формирования и анализа требований показана на рис. 3.2. Каждая организация использует собственный вариант этой модели, зависящий от «местных факторов»: опыта работы коллектива разработчиков, типа разрабатываемой системы, используемых стандартов и т.д.

Процесс формирования и анализа требований проходит через ряд этапов. Ими являются:

1. *Анализ предметной области.* Аналитики должны изучить предметную область, где будет эксплуатироваться система.
2. *Сбор требований.* Это – процесс взаимодействия с лицами, формирующими требования. Во время этого процесса продолжается анализ предметной области.
3. *Классификация требований.* На этом этапе бесформенный набор требований преобразуется в логически связанные группы требований.

4. *Разрешение противоречий.* Без сомнения, требования многочисленных лиц, занятых в процессе формирования требований, будут противоречивыми. На этом этапе определяются и разрешаются противоречия различного рода.

5. *Назначение приоритетов.* В любом наборе требований одни из них будут более важны, чем другие. На этом этапе совместно с лицами, формирующими требования, определяются наиболее важные требования.

6. *Проверка требований.* На этом этапе определяются их полнота, последовательность и непротиворечивость.

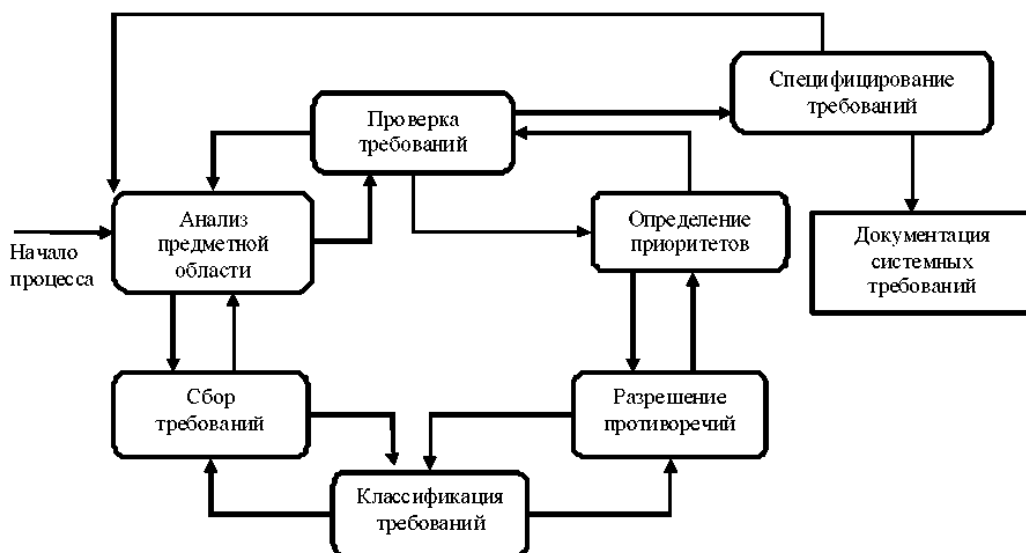


Рис. 3.2. Процесс формирования и анализа требований

Процесс формирования и анализа требований циклический, с обратной связью от одного этапа к другому. Цикл начинается с анализа предметной области и заканчивается проверкой требований. Понимание требований предметной области увеличивается в каждом цикле процесса формирования требований.

Рассмотрим три основных подхода к формированию требований: метод, основанный на множестве опорных точек зрения, сценарии и этнографический метод.

Опорные точки зрения

Подход с использованием различных *опорных* точек зрения к разработке требований признает различные (опорные) точки зрения на проблему и использует их в качестве основы построения и организации процесса формирования требований.

Различные методы предлагают разные трактовки выражения «точка зрения». Точки зрения можно трактовать следующим образом:

1. *Как источник информации о системных данных.* В этом случае на основе опорных точек зрения строится модель создания и использования данных в системе. В процессе формирования требований отбираются все такие точки

зрения (и на их основе определяются данные), которые будут созданы или использованы при работе системы, а также способы обработки этих данных.

2. *Как структуру представлений.* В этом случае точки зрения рассматриваются как особая часть модели системы. Например, на основе различных точек зрения могут разрабатываться модели "сущность-связь", модели конечного автомата и т.д.

3. *Как получатели системных сервисов.* В этом случае точки зрения являются внешними (относительно системы) получателями системных сервисов. Точки зрения помогают определить данные, необходимые для выполнения системных сервисов или их управления.

Наиболее эффективным подходом к анализу систем является использование внешних опорных точек зрения. На основе этого подхода разработан метод VORD (Viewpoint-Oriented Requirements Definition - определение требований на основе точек зрения) для формирования и анализа требований.

Основные этапы метода VORD (рис. 3.3) следующие:

1. Идентификация точек зрения, получающих системные сервисы, и идентификация сервисов, соответствующих каждой точке зрения.

2. Структурирование точек зрения - создание иерархии сгруппированных точек зрения. Общесистемные сервисы предоставляются более высоким уровням иерархии и наследуются точками зрения низшего уровня.

3. Документирование опорных точек зрения, которое заключается в точном описании идентифицированных точек зрения и сервисов.

4. Отображение системы точек зрения, которая показывает системные объекты, определенные на основе информации, заключенной в опорных точках зрения.



Рис. 3.3. Метод VORD

Аттестация требований

Аттестация требований должна продемонстрировать, что они действительно определяют ту систему, которую хочет иметь заказчик. Проверка требований важна, так как ошибки в спецификации требований могут привести к переделке системы и большим затратам, если будут обнаружены во время процесса разработки системы или после введения ее в эксплуатацию. Стоимость внесения в систему изменений, необходимых для устранения ошибок в требованиях, намного выше, чем исправление ошибок проектирования или кодирования. Причина в том, что изменение требований обычно влечет за собой значительные изменения в системе, после внесения которых она должна пройти повторное тестирование.

Во время процесса аттестации должны быть выполнены различные типы проверок требований:

1. *Проверка правильности требований.* Пользователь может считать, что система необходима для выполнения некоторых определенных функций. Однако дальнейшие размышления и анализ могут привести к необходимости введения дополнительных или новых функций. Системы предназначены для разных пользователей с различными потребностями, и поэтому набор требований будет представлять собой некоторый компромисс между требованиями пользователей системы.

2. *Проверка на непротиворечивость.* Спецификация требований не должна содержать противоречий. Это означает, что в требованиях не должно быть противоречащих друг другу ограничений или различных описаний одной и той же системной функции.

3. *Проверка на полноту.* Спецификация требований должна содержать требования, которые определяют все системные функции и ограничения, налагаемые на систему.

4. *Проверка на выполнимость.* На основе знания существующих технологий требования должны быть проверены на возможность их реального выполнения. Здесь также проверяются возможности финансирования и график разработки системы.

Существует ряд методов аттестации требований, которые можно использовать совместно или каждый в отдельности:

1. *Обзор требований.* Требования системно анализируются рецензентами.

2. *Прототипирование.* На этом этапе прототип системы демонстрируется конечным пользователям и заказчику. Они могут экспериментировать с этим прототипом, чтобы убедиться, что он отвечает их потребностям.

3. *Генерация тестовых сценариев.* В идеале требования должны быть такими, чтобы их реализацию можно было протестировать. Если тесты для требований разрабатываются как часть процесса аттестации, то часто это позволяет обнаружить проблемы в спецификации. Если такие тесты сложно или невозможно разработать, то обычно это означает, что требования трудно выполнить и поэтому необходимо их пересмотреть.

4. *Автоматизированный анализ непротиворечивости.* Если требования представлены в виде структурных или формальных системных моделей, можно использовать инструментальные CASE-средства для проверки непротиворечивости моделей. Для автоматизированной проверки непротиворечивости необходимо построить базу данных требований и затем проверить все требования в этой базе данных. Анализатор требований готовит отчет обо всех обнаруженных противоречиях.

Пользовательские и системные требования

На основании полученных моделей строятся пользовательские требования, т.е. описание на естественном языке функций, выполняемых системой, и ограничений, накладываемых на нее.

Пользовательские требования должны описывать внешнее поведение системы, основные функции и сервисы, предоставляемые системой, ее нефункциональные свойства. Необходимо выделить опорные точки зрения и сгруппировать требования в соответствии с ними. Пользовательские требования можно оформить как простым перечислением, так и используя нотацию вариантов использования.

Экспериментальная часть занятия

Пример выполнения задания

Рассмотрим использование метода VORD на первых трех шагах анализа требований для системы поддержки заказа и учета товаров. Для каждого товара фиксируется место хранения, количество товара и его поставщик. Система поддержки заказа и учета товаров должна обеспечивать добавление информации о новом товаре, изменение или удаление информации об имеющемся товаре, хранение (добавление, изменение и удаление) информации о поставщиках, включающей в себя название фирмы, ее адрес и телефон. При помощи системы составляются заказы поставщикам. Каждый заказ может содержать несколько позиций, в каждой позиции указываются наименование товара и его количество в заказе. Система по требованию пользователя формирует и выдает на печать следующую справочную информацию:

- список всех товаров;
- список товаров, имеющихся в наличии;
- список товаров, количество которых необходимо пополнить;
- список товаров, поставляемых данным поставщиком.

Первым шагом в формировании требований является идентификация опорных точек зрения. Во всех методах формирования требований, основанных на использовании точек зрения, начальная идентификация является наиболее трудной задачей.

Один из подходов к идентификации точек зрения – метод «мозгового штурма», когда определяются потенциальные системные сервисы и организации, взаимодействующие с системой. Организуется встреча лиц, участвующих в формировании требований, которые предлагают свои точки зрения. Эти точки зрения представляются в виде диаграммы, состоящей из ряда круговых областей, отображающих возможные точки зрения (рис. 3.4). Во время обсуждения необходимо идентифицировать потенциальные опорные точки зрения, системные сервисы, входные данные, нефункциональные требования, управляющие события и исключительные ситуации.

Следующей стадией процесса формирования требований будет идентификация опорных точек зрения (на рис. 3.4 показаны в виде темных круговых

областей) и сервисов (показаны в виде затененных областей). Сервисы должны соответствовать опорным точкам зрения. Но могут быть сервисы, которые не поставлены с ними в соответствие. Это означает, что на начальном этапе обсуждения некоторые опорные точки зрения не были идентифицированы.

В табл. 3.1 показано распределение сервисов для некоторых идентифицированных точек зрения. Один и тот же сервис может быть соотнесен с несколькими точками зрения.



Рис. 3.4. Диаграмма идентификации точек зрения

Таблица 3.1

Сервисы, соотнесенные с точками зрения

Клиент	Проверка наличия товара Покупка товара Получение чека Заказ товара Занесение покупателя и суммы покупки в базу данных
Покупатель	Занесение в список постоянных клиентов
Постоянный покупатель	Получение скидки Получение информации новых поступлений
Товар	Прием товара Занесение в базу данных (данные о поставщике, количестве, месте хранения и т.д.) Назначение цены Переопределение цены «Покупаемый» или «не покупаемый» товар
Поставщик	Занесение в базу данных (название, адрес, телефон и т.д.)
Продавец	Продажа товара Печать чека Доступ к каталогу Проверка наличия товара Оформление заказа покупателю

Сервисы, соотнесенные с точками зрения

Администратор	Доступ к базе данных Проверка статистики Переопределение цены Оформление заказа поставщику Печать заказа
---------------	--

Информация, извлеченная из точек зрения, используется для заполнения форм шаблонов точек зрения и организации точек зрения в иерархию наследования. Это позволяет увидеть общие точки зрения и повторно использовать информацию в иерархии наследования. Сервисы, данные и управляющая информация наследуются подмножеством точек зрения.

На рис. 3.5 показана часть иерархии точек зрения для системы поддержки заказа и учета товаров.

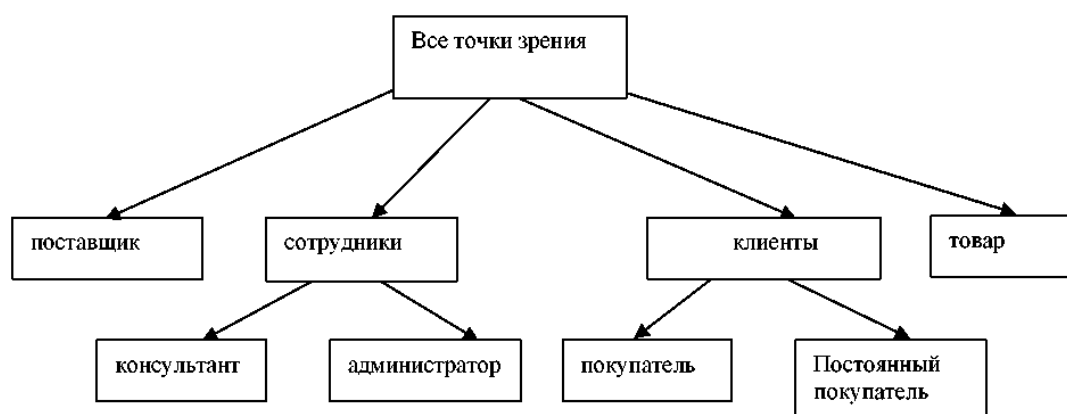


Рис. 3.5. Иерархия точек зрения

Требования к оформлению отчета**1. Требования к оформлению технического задания:**

1.1. Техническое задание оформляют в соответствии с ГОСТ 19.201-78 на листах формата А4 и А3, как правило, без заполнения полей листа. Номера листов (страниц) проставляют в верхней части листа над текстом.

1.2. Лист утверждения и титульный лист оформляют в соответствии с ГОСТ 2.610-2006. Информационную часть (аннотацию и содержание), лист регистрации изменений допускается в документ не включать.

1.3. Для внесения изменений и дополнений в техническое задание на последующих стадиях разработки программы или программного изделия к нему выпускают дополнение. Согласование и утверждение дополнения к техническому заданию проводят в том же порядке, который установлен для технического задания.

1.4. Техническое задание должно содержать следующие разделы:

- название программы и область применения;
- основание для разработки;

- назначение разработки;
- технические требования к программе или программному изделию;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки;
- приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

2. Требования к содержанию разделов:

2.1. В разделе «Наименование и область применения» указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

2.2. В разделе «Основание для разработки» должны быть указаны:

- документ (документы), на основании которых ведется разработка;
- организация, утвердившая этот документ, и дата его утверждения;
- наименование и (или) условное обозначение темы разработки.

2.3. В разделе «Назначение разработки» должно быть указано функциональное и эксплуатационное назначение программы или программного изделия.

2.4. Раздел «Технические требования к программе или программному изделию» должен содержать следующие подразделы:

- требования к функциональным характеристикам;
- требования к надежности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к маркировке и упаковке;
- требования к транспортированию и хранению;
- специальные требования.

2.4.1. В подразделе «Требования к функциональным характеристикам» должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам и т.п.

2.4.2. В подразделе «Требования к надежности» должны быть указаны требования к обеспечению надежного функционирования (обеспечение устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т.п.).

2.4.3. В подразделе «Условия эксплуатации» должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

2.4.4. В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их технических характеристик.

2.4.5. В подразделе «Требования к информационной и программной совместимости» должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования. При необходимости должна обеспечиваться защита информации и программ.

2.4.6. В подразделе «Требования к маркировке и упаковке» в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

2.4.7. В подразделе «Требования к транспортированию и хранению» должны быть указаны для программного изделия условия транспортирования, места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

2.5. В разделе «Технико-экономические показатели» должны быть указаны: ориентировочная экономическая эффективность предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.

2.6. В разделе «Стадии и этапы разработки» устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а также, как правило, сроки разработки и определяют исполнителей.

2.7. В разделе «Порядок контроля и приемки» должны быть указаны виды испытаний и общие требования к приемке работы.

2.8. В приложениях к техническому заданию при необходимости приводят:

- перечень научно-исследовательских и других работ, обосновывающих разработку;
- схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые могут быть использованы при разработке;
- другие источники разработки.

Контрольные вопросы

1. Объясните, почему требования, сформулированные разными лицами, могут быть противоречивыми?
2. Кто должен проводить обзор требований?
3. Назовите основные группы характеристик программного обеспечения в соответствии с техническим заданием.

Практическое занятие № 4

ТЕМА: « МЕТОДОЛОГИЯ ФУНКЦИОНАЛЬНОГО МОДЕЛИРОВАНИЯ IDEF0. ПОСТРОЕНИЕ КОНТЕКСТНОЙ ДИАГРАММЫ И ДЕКОМПОЗИЦИИ»

Цель занятия: изучить методологии функционального моделирования IDEF0.

Задачи:

1. Изучить основные методологии функционального моделирования SADT.
2. Разработать контекстную диаграмму и диаграмму декомпозиции IDEF0 на основе точек зрения информационной системы.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Построение контекстной диаграммы с помощью методологии IDEF0.
3. Выполнение декомпозиции на 1–2 уровня IDEF0 для определения основных информационных процессов функциональных задач.

Теоретические основы занятия

Построение контекстной диаграммы

Общая методология SADT (Structured Analysis and Design Technique, методология Росса) – методология структурного анализа и проектирования – состоит из трех частных методологий моделирования, основанных на графическом представлении системы разными языковыми средствами:

- IDEF0, которая используется для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающих между собой эти функции.
- IDEF1, которая применяется для построения информационной модели, отображающей структуру и содержание информационных потоков, необходимых для поддержки функций системы.
- IDEF2, которая позволяет построить динамическую модель изменения во времени функций, информационных потоков и ресурсов системы.

Далее излагается концепция моделирования в терминах языка IDEF0.


Модель SADT дает полное, точное и адекватное описание системы, имеющей конкретное назначение. Это назначение называется целью модели (purpose) и вытекает из ее формального определения. Формулировка цели выражает причину создания модели, т.е. содержит перечень вопросов, на которые должна отвечать модель, что в значительной мере определяет ее структуру.

Точка зрения – это указание на должностное лицо, с позиции которого разрабатывается модель. У модели может быть только одна точка зрения. Изменение точки зрения приводит к рассмотрению других аспектов объекта.

Аспекты, важные с одной точки зрения на этот объект, могут не появиться в модели, разрабатываемой с другой точки зрения.

Наиболее важные свойства объекта обычно выявляются на верхних уровнях иерархии, по мере декомпозиции функции верхнего уровня и разбиения ее на подфункции эти свойства уточняются. Каждая подфункция декомпозируется на элементы следующего уровня. Декомпозиция происходит до тех пор, пока не будет получена релевантная структура, позволяющая ответить на вопросы, сформулированные в цели моделирования.

Проектирование автоматизированных информационных систем может быть произведено с помощью CASE-средства BPwin, поддерживающего нотации IDEF0 (функциональная модель), IDEF3 (WorkFlow Diagram) и DFD (DataFlow Diagram). Функциональная модель предназначена как для описания существующих процессов (AS-IS), так и вновь проектируемых (TO-BE).










При запуске BPwin по умолчанию появляется основная панель инструментов, палитра инструментов (вид которой зависит от выбранной нотации) и, в левой части, навигатор модели – *Model Explorer*  (рис. 4.1).

Функциональность панели инструментов доступна из основного меню BPwin (табл. 4.1).

При создании новой модели возникает диалог, в котором следует указать, будет ли создана модель заново или она будет открыта из файла либо из репозитория *ModelMart*, внести имя модели и выбрать методологию, в которой будет построена модель (рис. 4.2). Система *ModelMart* – это хранилище моделей, к которому открыт доступ для участников проекта создания информационной системы.

Таблица 4.1

Описание элементов управления основной панели инструментов BPwin

Элемент управления	Описание	Пункт меню
	Создать новую модель	<i>File/New</i>
	Открыть модель	<i>File/Open</i>
	Сохранить модель	<i>File/Save</i>
	Напечатать модель	<i>File/Print</i>
	Выбор масштаба	<i>View/Zoom</i>
	Масштабирование	<i>View/Zoom</i>
	Проверка правописания	<i>Tools/Spelling</i>
	Включение и выключение навигатора модели <i>Model Explorer</i>	<i>View/Model Explorer</i>
	Включение и выключение дополнительной панели инструментов работы с <i>ModelMart</i>	<i>ModelMart</i>

BPwin поддерживает три методологии – *IDEF0*, *IDEF3* и *DFD*, каждая из которых решает свои специфические задачи. В BPwin возможно построение смешанных моделей, т.е. модель может содержать одновременно как диаграммы

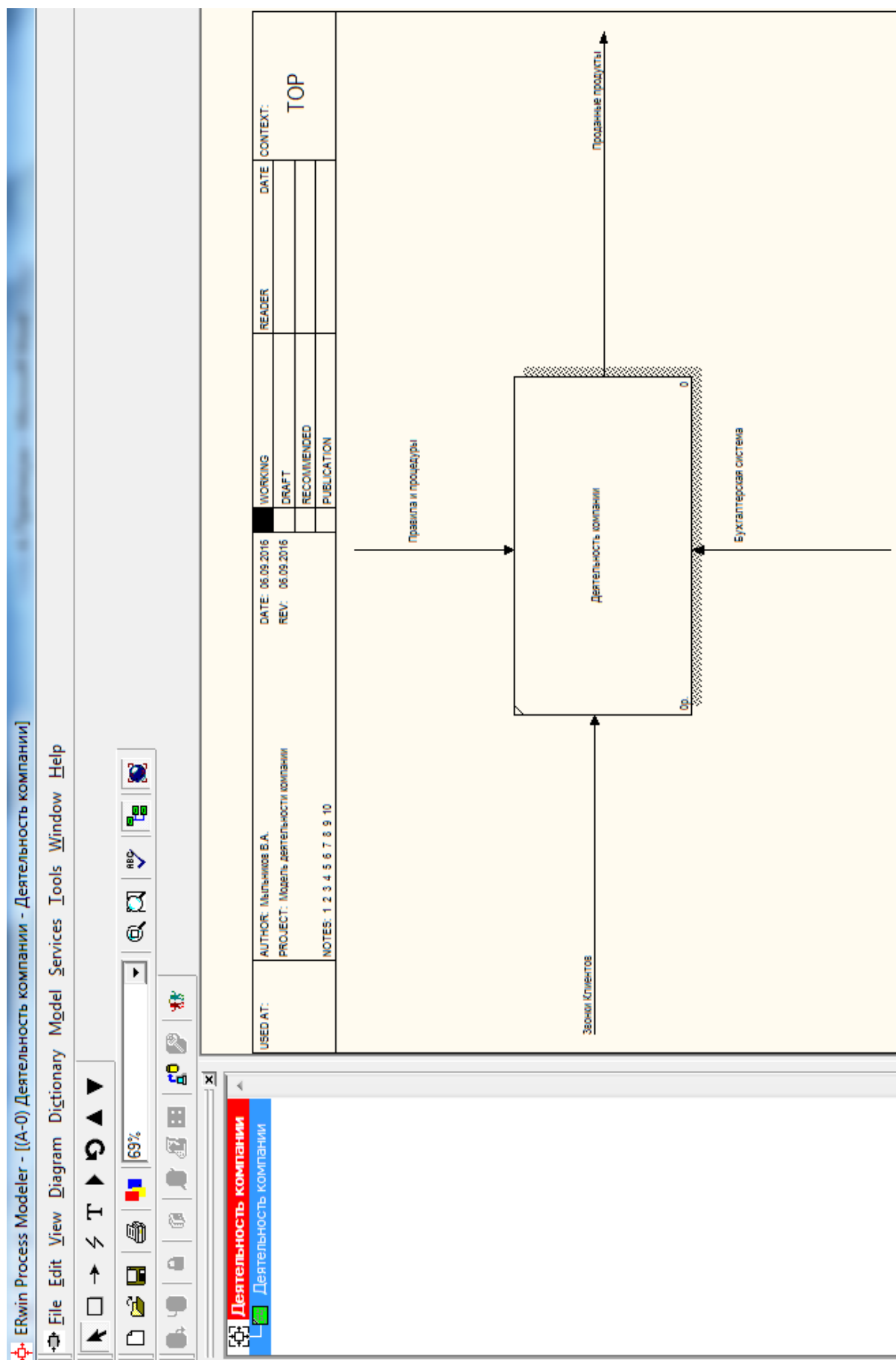


Рис. 4.1. Интегрированная среда разработки модели BPwin

IDEF0, так и *IDEF3* и *DFD*. Состав палитры инструментов изменяется автоматически, когда происходит переключение с одной нотации на другую.

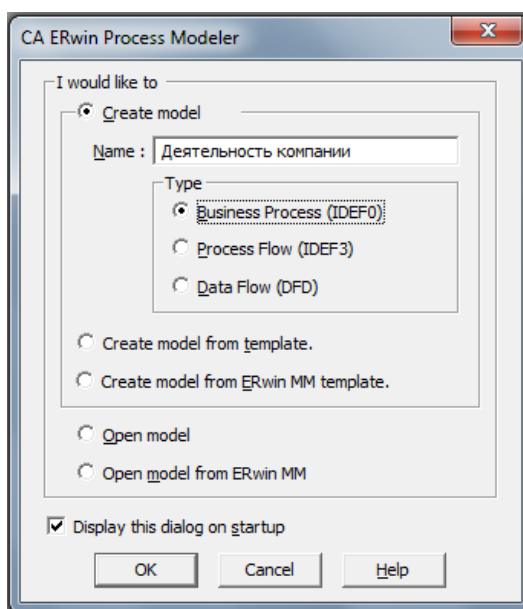


Рис. 4.2. Диалог создания модели

Пункты контекстного меню *Font Editor* и *Color Editor* вызывают соответствующие диалоги для установки шрифта и цвета объекта. ВРwin позволяет установить шрифт по умолчанию для объектов определенного типа на диаграммах и в отчетах с помощью контекстного меню диаграммы «*Parent Diagram Text Font.../Parent Diagram Title Text Font...*». Для этого следует выбрать удобный шрифт (например, *Arial* или *Courier New*), из выпадающего окна *Scripts* выбрать «*Кириллица*» и активировать опцию «*Change all occurrences*». Для исправления ошибок отображения русских символов в словаре необходимо запустить сценарий внесения дополнительных записей в ветку реестра. Список должен содержать только используемые шрифты с поправкой на русскую кодировку по умолчанию:

```
Windows Registry Editor Version 5.00
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows
NT\CurrentVersion\FontSubstitutes]
"Courier,0"="Courier New,204"
"Arial Cyr,0"="Arial,204"
"Courier New Cyr,0"="Courier New,204"
"Times New Roman Cyr,0"="Times New Roman,204"
"Arial"="Arial,204"
"Times New Roman"="Times New Roman,204"
"Courier New"="Courier New,204"
"Courier"="Courier New,204"
"Arial,0"="Arial,204"
"Courier New,0"="Courier New,204"
```

Наиболее удобным языком моделирования бизнес-процессов является *IDEF0*. Под моделью в *IDEF0* понимают описание системы (текстовое и графическое), которое должно дать ответ на некоторые заранее определенные вопросы. В

IDEF0 система представляется как совокупность взаимодействующих работ или функций. Такая чисто функциональная ориентация является принципиальной – функции системы анализируются независимо от объектов, которыми они оперируют, благодаря чему можно более четко смоделировать логику и взаимодействие процессов организации.

Моделируемая система рассматривается как произвольное подмножество неограниченного множества. Система имеет границу. Взаимодействие системы с окружающим миром описывается как вход (нечто, что перерабатывается системой), выход (результат деятельности системы), управление (стратегии и процедуры, под управлением которых производится работа) и механизм (ресурсы, необходимые для проведения работы). Находясь под управлением, система преобразует входы в выходы, используя механизмы.

Процесс моделирования какой-либо системы в *IDEF0* начинается с определения контекста, т.е. наиболее абстрактного уровня описания системы в целом. В контекст входит определение субъекта моделирования, цели и точки зрения на модель. Под субъектом понимается сама система. Описание области как системы в целом, так и ее компонентов является основой построения модели. Она должна быть в основном сформулирована изначально, поскольку именно область определяет направление моделирования и когда должна быть закончена модель. При формулировании области необходимо учитывать два компонента – широту и глубину. Широта подразумевает определение границ модели – определяется, что будет рассматриваться внутри системы, а что снаружи. Глубина определяет, на каком уровне детализации модель является завершенной. При определении глубины системы необходимо помнить об ограничениях времени – трудоемкость построения модели растет в геометрической прогрессии от глубины декомпозиции. После определения границ модели предполагается, что новые объекты не должны вноситься в моделируемую систему; поскольку все объекты модели взаимосвязаны, внесение нового объекта может быть не просто арифметической добавкой, но в состоянии изменить существующие взаимосвязи.

Цель моделирования (*Purpose*). Модель не может быть построена без четко сформулированной цели. Цель должна отвечать на следующие вопросы:

- Почему этот процесс должен быть смоделирован?
- Что должна показывать модель?
- Что может получить читатель?

Формулировка цели позволяет команде аналитиков сфокусировать усилия в нужном направлении.

Точка зрения (*Viewpoint*). Хотя при построении модели учитываются мнения различных людей, модель должна строиться с единой точки зрения. Точку зрения можно представить как взгляд человека, который видит систему в нужном для моделирования аспекте. Точка зрения должна соответствовать цели моделирования. Для этой цели обычно используют диаграммы *FEO* (*For Exposition Only*), которые будут описаны в дальнейшем.

IDEF0-модель предполагает наличие четко сформулированной цели, единственного субъекта моделирования и одной точки зрения. Для внесения

области, цели и точки зрения в модели *IDEF0* в BPwin следует выбрать пункт меню *Model/Model Properties*, вызывающий диалог *Model Properties* (рис. 4.3).

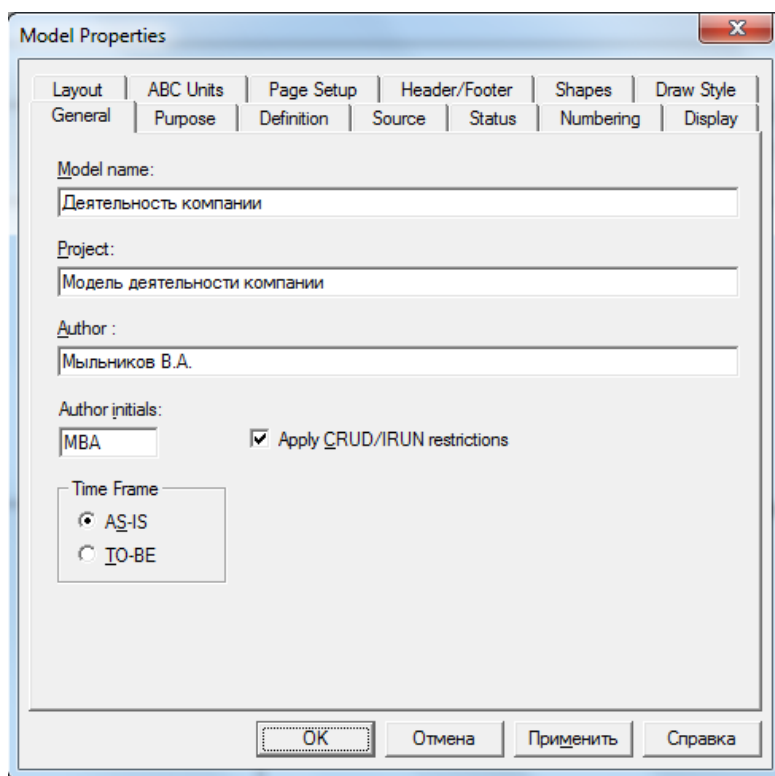


Рис. 4.3. Диалог *Model Properties*

В закладке *Status* того же диалога можно описать статус модели (черновой вариант, рабочий, окончательный и т.д.), время создания и последнего редактирования (отслеживается в дальнейшем автоматически по системной дате). В закладке *Source* описываются источники информации для построения модели (например, "Опрос экспертов предметной области и анализ документации"). Закладка *General* служит для внесения имени проекта и модели, имени и инициалов автора и временных рамок модели – *AS-IS* и *TO-BE*.

Модели *AS-IS* и *TO-BE*. Обычно сначала строится модель существующей организации работы – *AS-IS* (как есть). На основе модели *AS-IS* достигается консенсус между различными единицами бизнеса по тому, «кто что сделал» и что каждая единица бизнеса добавляет в процесс. Модель *AS-IS* позволяет выяснить, «что мы делаем сегодня» перед тем, как перейти на то, «что мы будем делать завтра». Анализ функциональной модели позволяет понять, где находятся наиболее слабые места, в чем будут состоять преимущества новых бизнес-процессов и насколько глубоким изменениям подвергнется существующая структура организации бизнеса. Найденные в модели *AS-IS* недостатки можно исправить при создании модели *TO-BE* (как будет) – модели новой организации бизнес-процессов. Модель *TO-BE* нужна для анализа альтернативных/лучших путей выполнения работы и документирования того, как компания будет делать бизнес в будущем.

Распространенная ошибка при создании модели *AS-IS* – это создание идеализированной модели. Примером может служить создание модели на основе знаний руководителя, а не конкретного исполнителя работ. Руководитель знаком с тем, как предполагается выполнение работы по руководствам и должностным инструкциям и часто не знает, как на самом деле подчиненные выполняют рутинные работы. В результате получается приукрашенная, искаженная модель, которая несет ложную информацию и которую невозможно в дальнейшем использовать для анализа. Такая модель называется *SHOULD_BE* (как должно бы быть).

Технология проектирования ИС подразумевает сначала создание модели *AS-IS*, ее анализ и улучшение бизнес-процессов, т.е. создание модели *TO-BE*, и только на основе модели *TO-BE* строятся модель данных, прототип и затем окончательный вариант ИС.

Основу методологии *IDEFO* составляет графический язык описания бизнес-процессов. Модель в нотации *IDEFO* представляет собой совокупность иерархически упорядоченных и взаимосвязанных диаграмм. Каждая диаграмма является единицей описания системы и располагается на отдельном листе.

Модель может содержать четыре типа диаграмм:

- контекстную диаграмму (в каждой модели может быть только одна контекстная диаграмма);
- диаграммы декомпозиции;
- диаграммы дерева узлов;
- диаграммы только для экспозиции (*FEO*).

Контекстная диаграмма является вершиной древовидной структуры диаграмм и представляет собой самое общее описание системы и ее взаимодействия с внешней средой. После описания системы в целом проводится разбиение ее на крупные фрагменты. Этот процесс называется функциональной декомпозицией, а диаграммы, которые описывают каждый фрагмент и взаимодействие фрагментов, называются диаграммами декомпозиции. После декомпозиции контекстной диаграммы проводится декомпозиция каждого большого фрагмента системы на более мелкие и т.д. до достижения нужного уровня подробности описания. Синтаксис описания системы в целом и каждого ее фрагмента одинаков во всей модели.

Диаграмма дерева узлов показывает иерархическую зависимость работ, но не взаимосвязи между работами. Диаграмм деревьев узлов может быть в модели сколько угодно много, поскольку дерево может быть построено на произвольную глубину и не обязательно с корня.

Диаграммы для экспозиции (*FEO*) строятся для иллюстрации отдельных фрагментов модели, для иллюстрации альтернативной точки зрения, либо для специальных целей.

Работы (*Activity*). Работы обозначают поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. Работы изображаются в виде прямоугольников. Все работы должны быть названы и определены. Имя работы должно быть

выражено отглагольным существительным, обозначающим действие (например, «Изготовление детали», «Прием заказа» и т.д.).

Стрелки (Arrow). Взаимодействие работ с внешним миром и между собой описывается в виде стрелок. Стрелки представляют собой некую информацию и именуются существительными (например, «Заготовка», «Изделие», «Заказ»).

В *IDEF0* различают пять типов стрелок. Рассмотрим их.

Вход (Input) – материал или информация, которые используются или преобразуются работой для получения результата (выхода). Допускается, что работа может не иметь ни одной стрелки входа. Каждый тип стрелок подходит к определенной стороне прямоугольника, изображающего работу, или выходит из нее. Стрелка входа рисуется как входящая в левую грань работы.

Управление (Control) – правила, стратегии, процедуры или стандарты, которыми руководствуется работа. Каждая работа должна иметь хотя бы одну стрелку управления. Стрелка управления рисуется как входящая в верхнюю грань работы. Управление влияет на работу, но не преобразуется работой. Если цель работы - изменить процедуру или стратегию, то такая процедура или стратегия будет для работы входом. В случае возникновения неопределенности в статусе стрелки (управление или вход) рекомендуется рисовать стрелку управления.



Выход (Output) – 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 рисуется как исходящая из правой грани работы.

Механизм (Mechanism) – ресурсы, которые выполняют работу, например персонал предприятия, станки, устройства и т.д. Стрелка механизма рисуется как входящая в нижнюю грань работы. По усмотрению аналитика стрелки механизма могут не изображаться в модели.

Вызов (Call) – специальная стрелка, указывающая на другую модель работы. Стрелка вызова рисуется как исходящая из нижней грани работы. Стрелка вызова используется для указания того, что некоторая работа выполняется за пределами моделируемой системы. В *VRwin* стрелки вызова используются в механизме слияния и разделения моделей.

Граничные стрелки. 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.

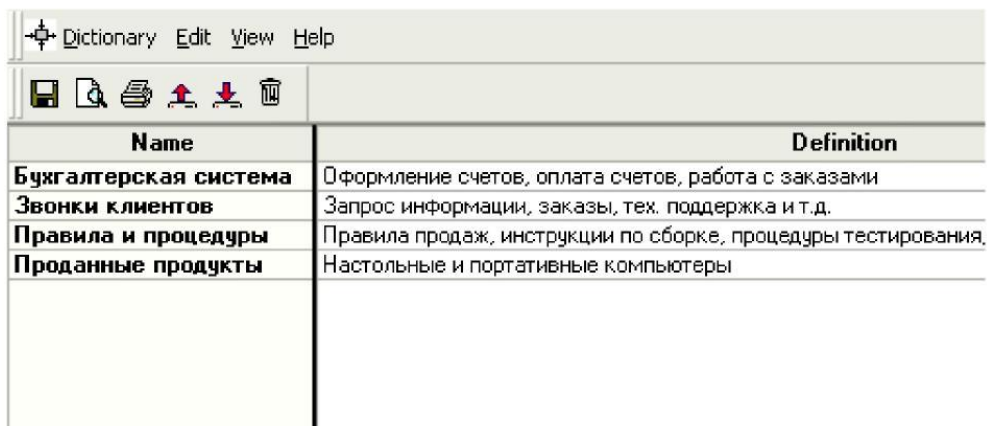
Для внесения граничной стрелки входа следует:

- щелкнуть по кнопке с символом стрелки  в палитре инструментов и переместить курсор в левую часть экрана, пока не появится черная полоска;
- щелкнуть один раз по полоске в левой стороне экрана (откуда выходит стрелка) и еще раз в левой части работы со стороны входа (где заканчивается стрелка);
- вернуться в палитру инструментов и выбрать опцию редактирования стрелки .

- щелкнуть правой кнопкой мыши на линии стрелки, в контекстном меню выбрать *Name* и ввести имя стрелки в закладке *Name* диалога *Arrow Properties*.

Стрелки управления, выхода, механизма и выхода изображаются аналогично. Для рисования стрелки выхода, например, следует щелкнуть по кнопке с символом стрелки в палитре инструментов, щелкнуть в правой части работы со стороны выхода (где начинается стрелка), перенести курсор к правой стороне экрана, пока не появится начальная штриховая полоска, и щелкнуть один раз по штриховой полоске.

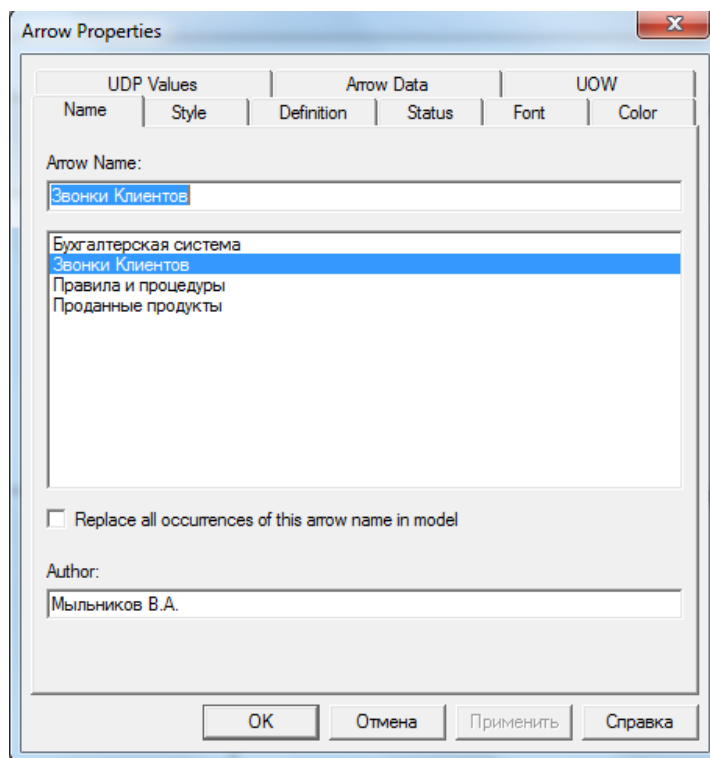
Имена вновь внесенных стрелок автоматически заносятся в словарь *Arrow Dictionary* (рис. 4.4).



Name	Definition
Бухгалтерская система	Оформление счетов, оплата счетов, работа с заказами
Звонки клиентов	Запрос информации, заказы, тех. поддержка и т.д.
Правила и процедуры	Правила продаж, инструкции по сборке, процедуры тестирования
Проданные продукты	Настольные и портативные компьютеры

Рис. 4.4. Словарь *Arrow Dictionary*

Изменить данные о стрелках можно с помощью меню *Arrow Properties* (рис. 4.5).



Arrow Properties

UDP Values | Arrow Data | UOW

Name | Style | Definition | Status | Font | Color

Arrow Name:

Звонки Клиентов

Бухгалтерская система

Звонки Клиентов

Правила и процедуры

Проданные продукты

☐ Replace all occurrences of this arrow name in model

Author:

Мыльников В.А.

OK Отмена Применить Справка

Рис. 4.5. Диалог *Arrow Properties*

Декомпозиция функциональных блоков

Диаграмма декомпозиции предназначена для детализации работы. В отличие от моделей, отображающих структуру организации, работа на диаграмме верхнего уровня в *IDEF0* – это не элемент управления нижестоящими работами. Работы нижнего уровня – это то же самое, что работы верхнего уровня, но в более детальном изложении. Как следствие этого границы работы верхнего уровня – это то же самое, что границы диаграммы декомпозиции.

ICOM (аббревиатура от *Input, Control, Output* и *Mechanism*) – это коды, предназначенные для идентификации граничных стрелок. Код *ICOM* содержит префикс, соответствующий типу стрелки (*I, C, O* или *M*), и порядковый номер.

ВРwin вносит *ICOM*-коды автоматически. Для отображения *ICOM*-кодов следует включить опцию *Show ICOM codes* на закладке *Display* диалога *Model Properties* (меню *Model/Model Properties*).

Чтобы не возникло неоднозначных трактовок, в словаре стрелок каждому понятию можно дать расширенное и, при необходимости, формальное определение.

Содержимое словаря стрелок можно распечатать в виде отчета (меню *Tools/Reports/Arrow Report*) и получить тем самым толковый словарь терминов предметной области, использующихся в модели.

Связи работ. При декомпозиции работы входящие в нее и исходящие из нее стрелки автоматически появляются на диаграмме декомпозиции (миграция стрелок), но при этом не касаются работ. Такие стрелки называются несвязанными (*unconnected border arrow*) и воспринимаются в ВРwin как синтаксическая ошибка.

Для связывания стрелок входа, управления или механизма необходимо перейти в режим редактирования стрелок, щелкнуть по наконечнику стрелки и щелкнуть по соответствующему сегменту работы. Для связывания стрелки выхода необходимо перейти в режим редактирования стрелок, щелкнуть по сегменту выхода работы и затем по стрелке.

Для связи работ между собой используются внутренние стрелки, т.е. стрелки, которые не касаются границы диаграммы, начинаются у одной и кончаются у другой работы. Для рисования внутренней стрелки необходимо в режиме рисования стрелок щелкнуть по сегменту (например, выхода) одной работы и затем по сегменту (например, входа) другой.

В *IDEF0* различают пять типов связей работ:

- *связь по входу (output-input)*, когда стрелка выхода вышестоящей работы (далее – просто выход) направляется на вход нижестоящей;
- *связь по управлению (output-control)*, 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;
- *обратную связь по входу (output-input feedback)*, когда выход нижестоящей работы направляется на вход вышестоящей. Такая связь, как правило, используется для описания циклов;

- *обратную связь по управлению (output-control feedback)*, когда выход нижестоящей работы направляется на управление вышестоящей. Обратная связь по управлению часто свидетельствует об эффективности бизнес-процесса;

- *связь выход-механизм (output-mechanism)*, 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.

Явные стрелки. Явная стрелка имеет источником одну единственную работу и назначением тоже одну единственную работу.

Разветвляющиеся и сливающиеся стрелки. Одни и те же данные или объекты, порожденные одной работой, могут использоваться сразу в нескольких других работах. С другой стороны, стрелки, порожденные в разных работах, могут представлять собой одинаковые или однородные данные или объекты, которые в дальнейшем используются или перерабатываются в одном месте. Для моделирования таких ситуаций в *IDEF0* используются разветвляющиеся и сливающиеся стрелки. Для разветвления стрелки нужно в режиме редактирования стрелки щелкнуть по фрагменту стрелки и по соответствующему сегменту работы. Для слияния двух стрелок выхода нужно в режиме редактирования стрелки сначала щелкнуть по сегменту выхода работы, а затем по соответствующему фрагменту стрелки.

Смысл разветвляющихся и сливающихся стрелок передается именованием каждой ветви стрелок. Существуют определенные правила именования таких стрелок. Рассмотрим их на примере разветвляющихся 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 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. *BPwin* определяет такую стрелку как синтаксическую ошибку.

Правила именования сливающихся стрелок полностью аналогичны - ошибкой будет считаться стрелка, которая после слияния не именована, а до слияния не именована какая-либо из ее ветвей. Для именования отдельной ветви разветвляющихся и сливающихся стрелок следует выделить на диаграмме только одну ветвь, после этого вызвать редактор имени и присвоить имя стрелке. Это имя будет соответствовать только выделенной ветви.

Работы (Activity). Работы обозначают поименованные процессы, функции или задачи, которые происходят в течение определенного времени и имеют распознаваемые результаты. Работы изображаются в виде прямоугольников. Все работы должны быть названы и определены. Имя работы должно быть выражено отглагольным существительным, обозначающим действие (например, «Изготовление детали», «Прием заказа» и т.д.). Работа «Изготовление детали» может иметь, например, следующее определение: работа относится к полному

циклу изготовления изделия от контроля качества сырья до отгрузки готового упакованного изделия. При создании новой модели (меню *File/New*) автоматически создается контекстная диаграмма с единственной работой, изображающей систему в целом.

Для внесения имени работы следует щелкнуть по работе правой кнопкой мыши, выбрать в меню *Name Editor* и в появившемся диалоге внести имя работы. Для описания других свойств работы служит диалог *Activity Properties*.

Диаграммы декомпозиции содержат родственные работы, т.е. дочерние работы, имеющие общую родительскую работу. Декомпозировать работу на одну работу не имеет смысла: диаграммы с количеством работ более восьми получаются перенасыщенными и плохо читаются. Для обеспечения наглядности и лучшего понимания моделируемых процессов рекомендуется использовать от трех до шести блоков на одной диаграмме.

Работы на диаграммах декомпозиции обычно располагаются по диагонали от левого верхнего угла к правому нижнему. Такой порядок называется порядком доминирования. Согласно этому принципу расположения в левом верхнем углу располагается самая важная работа или работа, выполняемая по времени первой. Далее вправо вниз располагаются менее важные или выполняемые позже работы. Такое расположение облегчает чтение диаграмм, кроме того, на нем основывается понятие взаимосвязей работ.

Каждая из работ на диаграмме декомпозиции может быть, в свою очередь, декомпозирована. На диаграмме декомпозиции работы нумеруются автоматически слева направо. Номер работы показывается в правом нижнем углу. В левом верхнем углу изображается небольшая диагональная черта, которая показывает, что данная работа не была декомпозирована.

Стрелки. Альтернативный метод внесения имен и свойств стрелок – использование словаря стрелок (вызов словаря с помощью меню *Dictionary/Arrow*). Если внести имя и свойства стрелки в словарь, то ее можно будет внести в диаграмму позже. Стрелку нельзя удалить из словаря, если она используется на какой-либо диаграмме. Если удалить стрелку из диаграммы, то из словаря она не удаляется. Имя и описание такой стрелки может быть использовано в дальнейшем. Для добавления стрелки необходимо перейти в конец списка и щелкнуть правой кнопкой по последней строке. Возникает новая строка, в которой нужно внести имя и свойства стрелки.

Экспериментальная часть занятия

Постановка задачи

В качестве примера рассматривается деятельность вымышленной компании. Компания занимается в основном сборкой и продажей настольных компьютеров и ноутбуков. Компания не производит компоненты самостоятельно, а только собирает и тестирует компьютеры.

Основные процедуры в компании таковы:


- продавцы принимают заказы клиентов;
- операторы группируют заказы по типам компьютеров;

- операторы собирают и тестируют компьютеры;
- операторы упаковывают компьютеры согласно заказам;
- кладовщик отгружает клиентам заказы.


Компания использует купленную бухгалтерскую информационную систему, которая позволяет оформить заказ, счет и отследить платежи по счетам.

Порядок выполнения работы

1. Запустите BPwin. Если появляется диалог «*ModelMart Connection Manager*», нажмите на кнопку *Cancel*.

2. Щелкните по кнопке . Появляется диалог *I would like to*. Внесите имя модели «Деятельность компании» и выберите Type – *IDEF*. Нажмите на кнопку *OK*.

Появляется меню *Properties for New Models*. Во вкладке *General* вводится фамилия и инициалы автора, остальные вкладки используются для определения настроек проекта.

3. Автоматически создается контекстная диаграмма. Обратите внимание на кнопку  на панели инструментов. Эта кнопка включает и выключает инструмент просмотра и навигации – *Model Explorer* (появляется слева). *Model Explorer* имеет три вкладки: *Activities*, *Diagrams* и *Objects*. Во вкладке *Activities* щелчок правой кнопкой по объекту позволяет редактировать его свойства. Вы можете вызвать справку с помощью клавиши *F1* или через меню *Help*.

4. Измените шрифт для правильного отображения русских букв. Зайдите в меню *Model/Default Fonts* и выберите *Parent Diagram Text*. Поставьте галочку *Change all occurrences* и нажмите *OK*. Если ничего не изменилось, повторите эту операцию с другим подменю, например: *Parent Diagram Title Text*.

5. Для изменений свойств модели используется меню *Model Properties*. По умолчанию тип модели: *Time Frame: AS-IS*. Во вкладке *Purpose* внесите цель – «Моделировать текущие (AS-IS) бизнес-процессы компании» и точку зрения – «Директор».

6. Во вкладке *Definition* внесите определение «Это учебная модель, описывающая деятельность компании» и цель *Scope*: «Общее управление бизнесом компании: исследование рынка, закупка компонентов, сборка, тестирование и продажа продуктов».

7. Перейдите на контекстную диаграмму и правой кнопкой мыши щелкните по работе. В контекстном меню выберите *Name*. Во вкладке *Name* внесите имя «Деятельность компании».

8. Во вкладке *Definition* внесите определение «Текущие бизнес-процессы компании».

9. Создайте стрелки на контекстной диаграмме с помощью меню *Model/Arrow Editor* согласно табл. 4.2.

Стрелки контекстной диаграммы

<i>Arrow Name</i>	<i>Arrow Definition</i>
Бухгалтерская система	Оформление счетов, оплата счетов, работа с заказами
Звонки клиентов	Запросы информации, заказы, техподдержка и т.д.
Правила и процедуры	Правила продаж, инструкции по сборке, процедуры тестирования, критерии производительности и т.д.
Проданные продукты	Настольные и портативные компьютеры

10. С помощью кнопки **T** внесите текст в поле диаграммы (рис. 4.6) – точку зрения и цель (см. рис. 4.8).

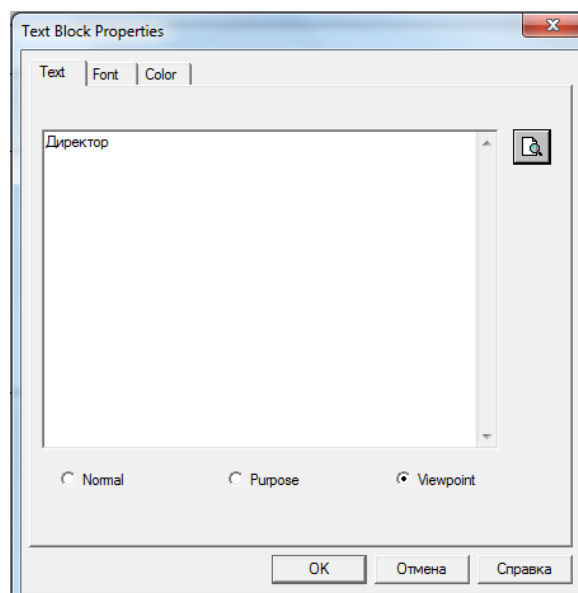


Рис. 4.6. Внесение текста в поле диаграммы с помощью редактора Text Block Editor

11. Создайте отчет по модели. Меню – *Tools/Reports/Model Report* (рис. 4.7).

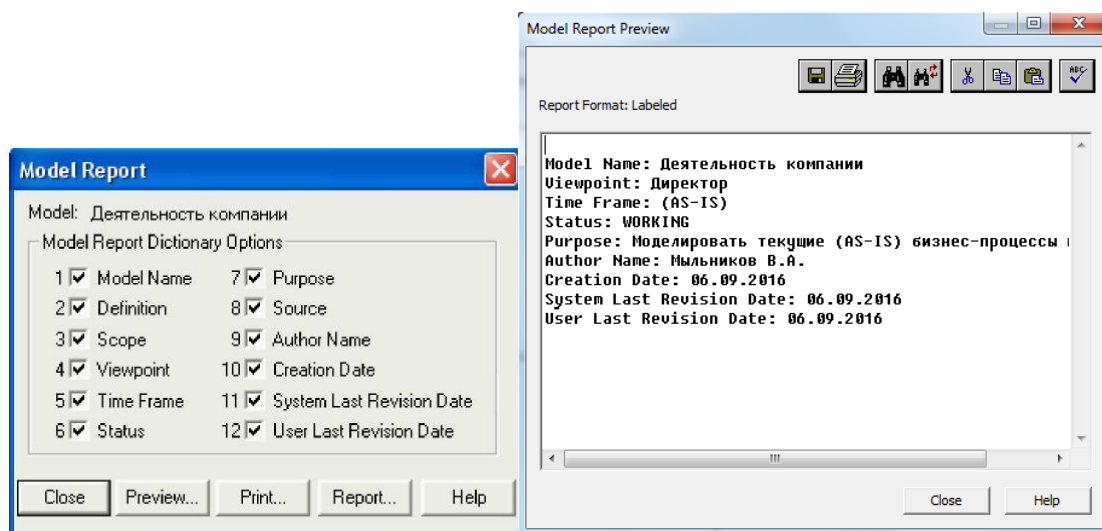


Рис. 4.7. Отчет Model Report

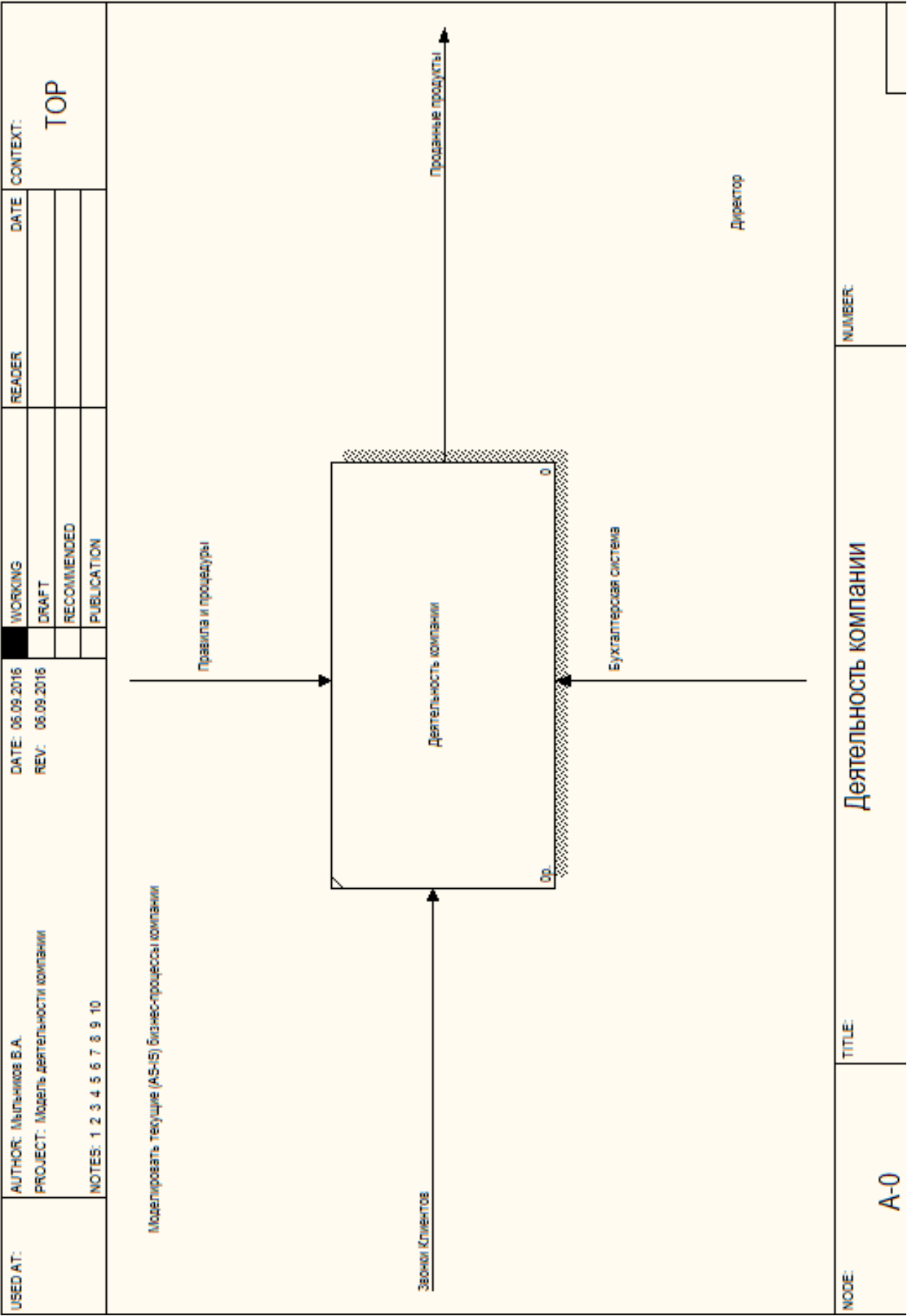


Рис. 4.8. Контекстная диаграмма

12. Выберите кнопку перехода на нижний уровень в палитре инструментов ▼ и в диалоге *Activity Box Count* установите число работ на диаграмме нижнего уровня, равное трем, и нажмите *OK* (рис. 4.9).

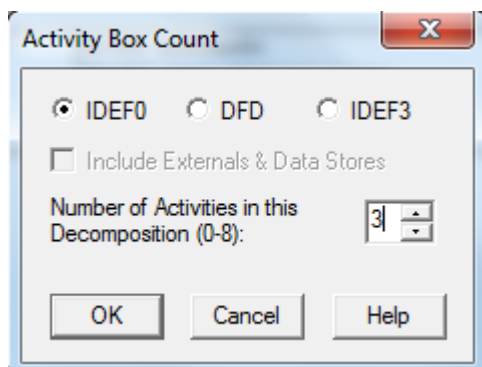


Рис. 4.9. Создание диаграммы декомпозиции

13. Автоматически будет создана диаграмма декомпозиции. Правой кнопкой мыши щелкните по работе, выберите *Name* и внесите имя работы. Повторите операцию для всех трех работ. Затем внесите определение, статус и источник для каждой работы согласно табл. 4.3.

Таблица 4.3


Работы диаграммы декомпозиции A0


<i>Activity Name</i>	<i>Definition</i>
Продажи и маркетинг	Телемаркетинг и презентации, выставки
Сборка и тестирование компьютеров	Сборка и тестирование настольных и портативных компьютеров
Отгрузка и получение	Отгрузка заказов клиентам и получение компонентов от поставщиков


14. Для изменения свойств работ после их внесения в диаграмму можно воспользоваться словарем работ. Вызов словаря – меню *Dictionary/Activity* (рис. 4.10).

Name	Definition
Деятельность компании	Текущие бизнес-процессы компании
Отгрузка и получение	Отгрузка заказов клиентам и получение компонентов
Продажи и маркетинг	Телемаркетинг и презентации, выставки
Сборка и	Сборка и тестирование настольных и портативных ком

Рис. 4.10. Словарь Activity Dictionary

15. Если описать имя и свойства работы в словаре, то ее можно будет внести в диаграмму позже с помощью кнопки  в палитре инструментов. Невозможно удалить работу из словаря, если она используется на какой-либо диаграмме. Если работа удаляется из диаграммы, то из словаря она не удаляется. Имя и описание такой работы может быть использовано в дальнейшем. Для добавления работы в словарь необходимо перейти в конец

списка и щелкнуть правой кнопкой по последней строке. Возникает новая строка, в которой нужно внести имя и свойства работы. Также можно воспользоваться клавишей табуляции для введения информации. Для удаления всех имен работ, не использующихся в модели, щелкните по кнопке  (Purge).

16. Измените стрелку «Проданные продукты» и «Звонки клиентов». Перейдите в режим рисования стрелок. Свяжите граничные стрелки (кнопка  на палитре инструментов) так, как показано на рис. 4.11.

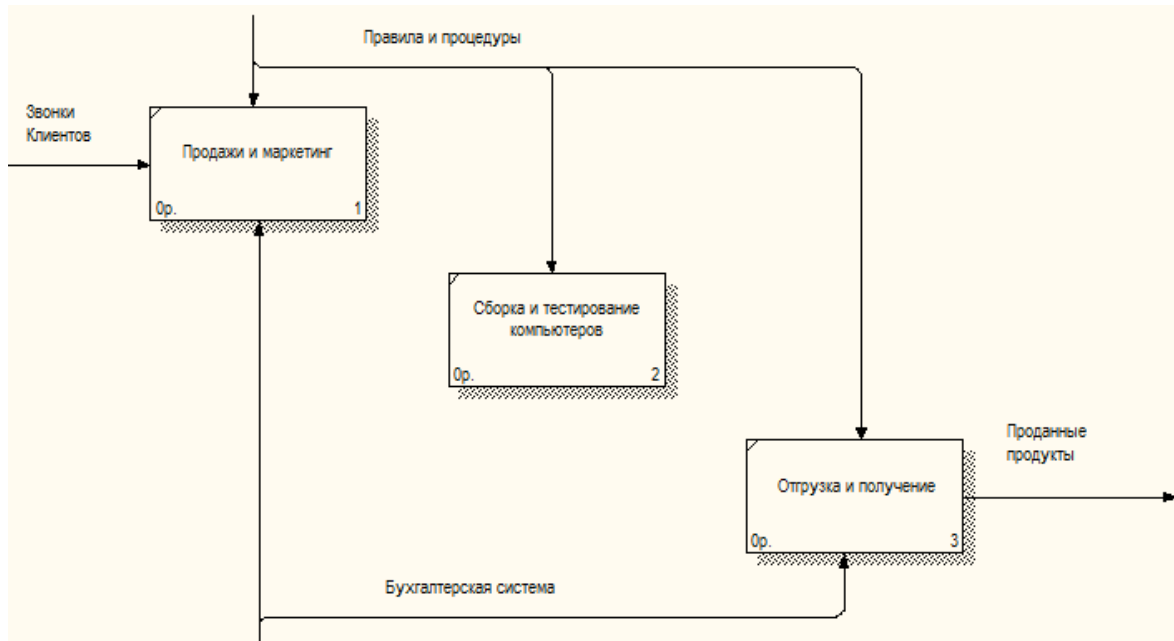


Рис. 4.11. Связанные граничные стрелки на диаграмме A0

17. Внесите определение для ветви «Бухгалтерская система»: «Инструкции по сборке», «Процедуры тестирования», «Критерии производительности» и т.д. Правой кнопкой мыши щелкните по ветви «Бухгалтерская система», переименуйте ее в «Система оформления заказов».

18. Создайте новые внутренние стрелки «Заказы клиентов», «Собранные компьютеры» и «Неисправные компоненты» так, как показано на рис. 4.12. Дайте название «Правила сборки и тестирования» стрелке управления, входящую в работу «Сборка и тестирование компьютеров».

19. Создайте стрелку обратной связи (по управлению) «Результаты сборки и тестирования», идущую от работы, «Сборка и тестирование компьютеров» к работе «Продажи и маркетинг». Методом *drag&drop* перенесите имена стрелок так, чтобы их было удобнее читать. Если необходимо, установите *Squiggle* (из контекстного меню). Результат изменений показан на рис. 4.13.

20. Свяжите работу «Продажи и маркетинг» с граничной стрелкой выхода «Маркетинговые материалы». На уровне работы «Сборка и тестирование компьютеров» для стрелки «Маркетинговые материалы» выберите опцию *Trim* из контекстного меню в целях уменьшения длины стрелки. Результат выполнения работы показан на рис. 4.14.

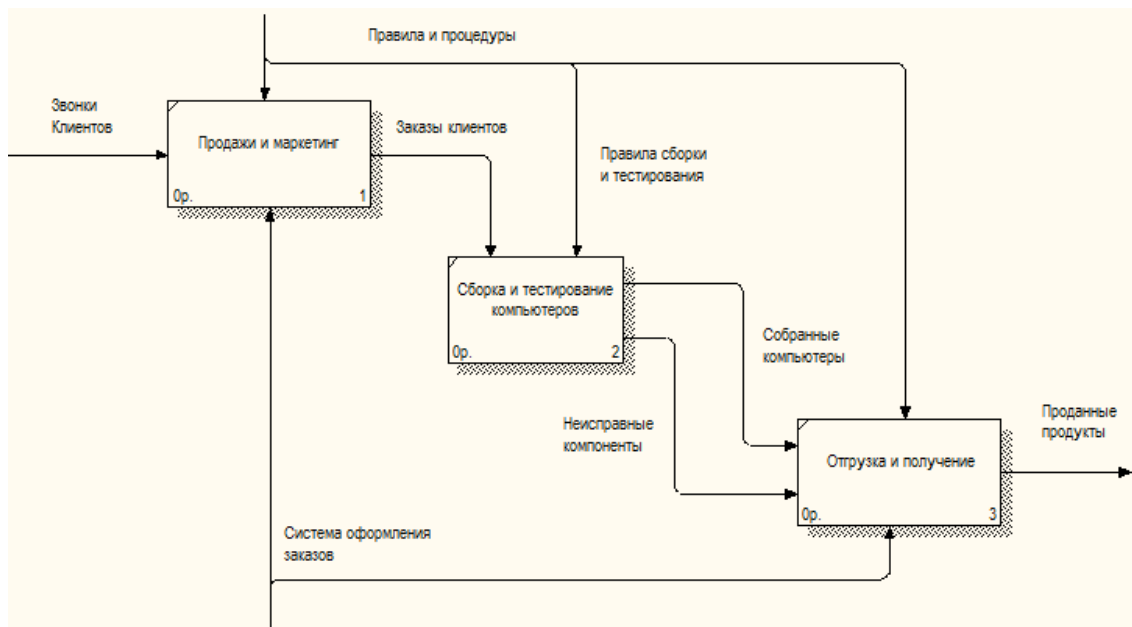


Рис. 4.12. Внутренние стрелки диаграммы АО

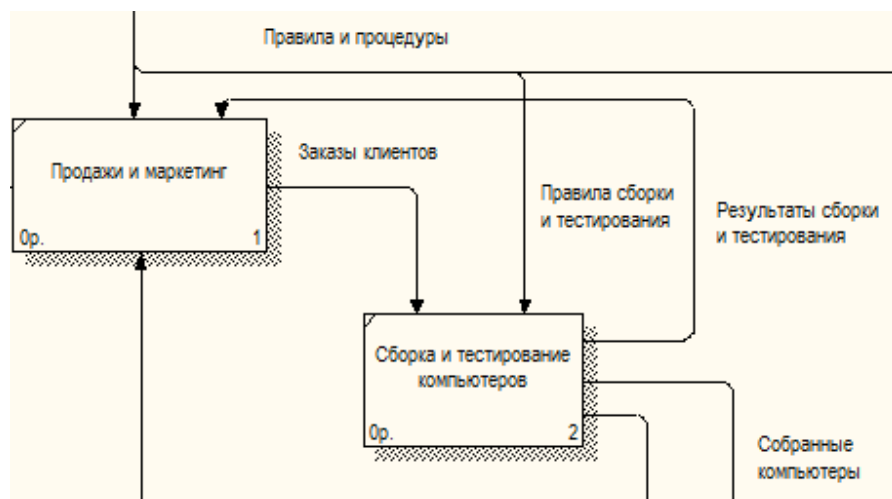


Рис. 4.13. Результат выполнения задания 6

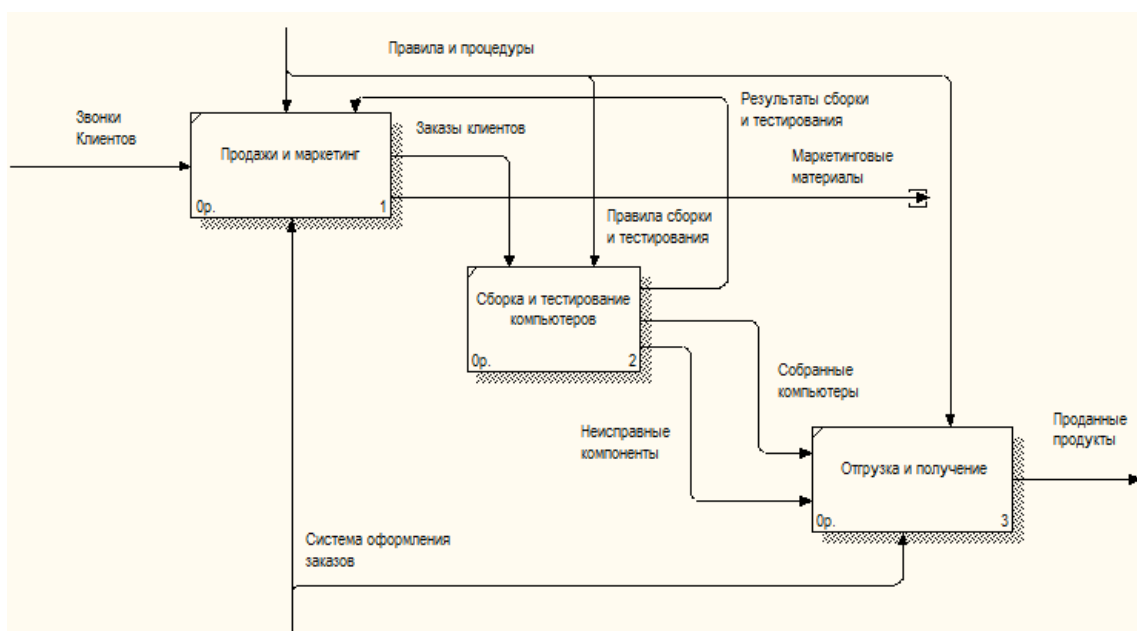


Рис. 4.14. Результат выполнения – диаграмма АО

21. Работа по продажам и маркетингу заключается в ответах на телефонные звонки клиентов, предоставлении клиентам информации о ценах, оформлении заказов, внесении заказов в информационную систему и исследовании рынка. На основе этой информации декомпозируйте работу «Продажи и маркетинг».

22. Создайте работы «Предоставление информации о ценах», «Оформление заказов» и «Исследование рынка».

23. Убедитесь, что все стрелки модели связаны, если необходимо перерисуйте несвязанные стрелки. Результат декомпозиции представлен на рис. 4.15.

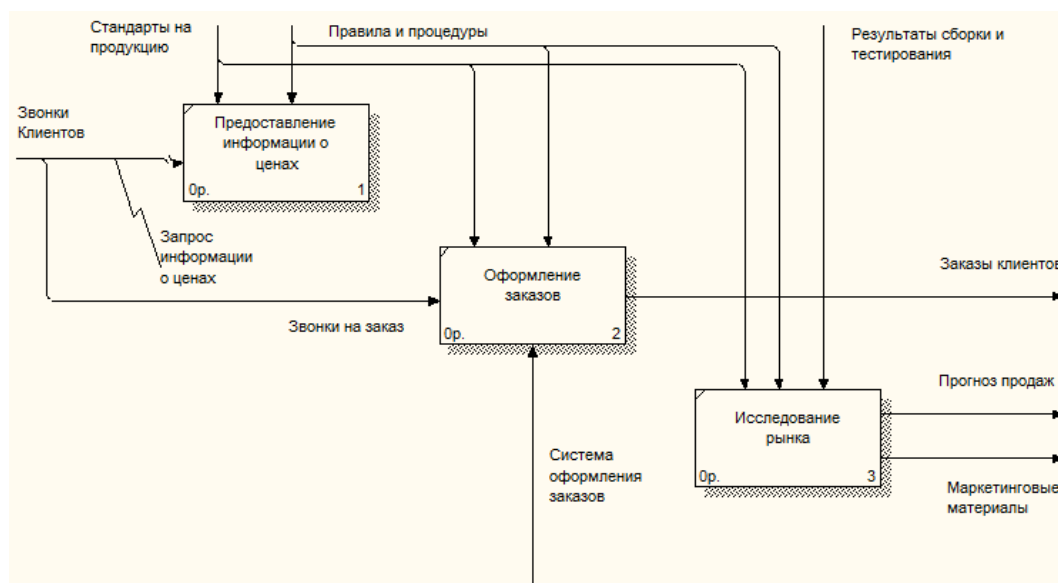


Рис. 4.15. Диаграмма A2 «Продажи и маркетинг»

24. Выполните туннелирование стрелки «Прогноз продаж» на вышестоящий уровень, а стрелки «Стандарты на продукцию» до контекстной диаграммы.

25. На диаграмме A0 «Деятельность компании» переместите выход стрелки «Прогноз продаж» на вход по управлению работы «Сборка и тестирование компьютеров», перейдите на диаграмму декомпозиции этой работы и соедините данную стрелку по управлению с работой «Отслеживание расписания и управление сборкой и тестированием».

Контрольные вопросы

1. Определите назначение методологии IDEF0.
2. Объясните правила изображения функциональных блоков в IDEF0.
3. Что такое ICOM-коды?
4. Какие бывают типы стрелок?
5. Что такое словарь работ, стрелок?
6. Какие бывают типы связей работ?
7. Что называется декомпозицией в методологии IDEF0?

Практическое занятие № 5

ТЕМА: «ТУННЕЛИРОВАНИЕ СТРЕЛОК. РАСЩЕПЛЕНИЕ И СЛИЯНИЕ МОДЕЛИ ПРИ ГРУППОВОЙ РАБОТЕ»

Цель занятия: изучить механизм туннелирования стрелок в диаграммах декомпозиций и навыки групповой работы при расщеплении и слиянии модели.

Задачи:

1. Изучить механизм туннелирования стрелок.
2. Изучить механизм расщепления и слияния модели.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Построение контекстной диаграммы с помощью методологии IDEF0.
3. Выполнение декомпозиции на 1–2 уровня IDEF0 для определения основных информационных процессов функциональных задач.

Теоретические основы занятия

В реальных диаграммах к каждой работе может подходить и от каждой может отходить около десятка стрелок. Если диаграмма содержит 6–8 работ, то она может содержать 30–40 стрелок, причем они могут сливаться, разветвляться и пресекаться. Такие диаграммы могут стать очень плохо читаемыми. В *IDEF0* существуют соглашения по рисованию диаграмм, которые призваны облегчить чтение и экспертизу модели. Некоторые из этих правил *BPwin* поддерживает автоматически, выполнение других следует обеспечить вручную.

Прямоугольники работ должны располагаться по диагонали с левого верхнего в правый нижний угол (порядок доминирования). При создании новой диаграммы декомпозиции *BPwin* автоматически располагает работы именно в таком порядке. В дальнейшем можно добавить новые работы или изменить расположение существующих, но нарушать диагональное расположение работ по возможности не следует. Порядок доминирования подчеркивает взаимосвязь работ, позволяет минимизировать изгибы и пересечения стрелок.

Следует максимально увеличивать расстояние между входящими или выходящими стрелками на одной грани работы. Если включить опцию *Line Drawing: Automatically space arrows* на закладке *Layout* диалога *Model Properties* (меню *Edit/Model Properties*), то *BPwin* будет располагать стрелки нужным образом автоматически.

Следует максимально увеличить расстояние между работами, поворотами и пересечениями стрелок.

Если две стрелки проходят параллельно, начинаются из одной и той же грани одной работы и заканчиваются на одной и той же грани другой работы, то по возможности следует их объединить и назвать единым термином.

Обратные связи по входу рисуются нижней петлей, обратная связь по управлению – верхней. ВРwin автоматически рисует обратные связи нужным образом.

Циклические обратные связи следует рисовать только в случае крайней необходимости, когда подчеркивают значение повторно используемого объекта. Принято изображать такие связи на диаграмме декомпозиции. ВРwin не позволяет создать циклическую обратную связь за один прием. Если все же необходимо изобразить такую связь, то следует сначала создать обычную связь по выходу, затем разветвить стрелку, направить новую, ветвь обратно ко входу работы источника и, наконец, удалить старую ветвь стрелки выхода (рис. 5.1).

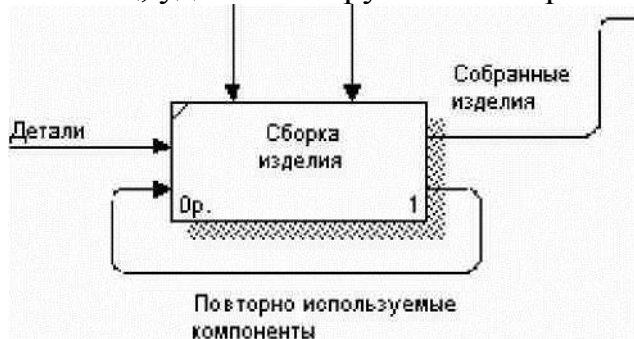


Рис. 5.1. Пример обратной циклической связи

Следует минимизировать число пересечений, петель и поворотов стрелок. Это ручная и, в случае насыщенных диаграмм, творческая работа (рис. 5.2).

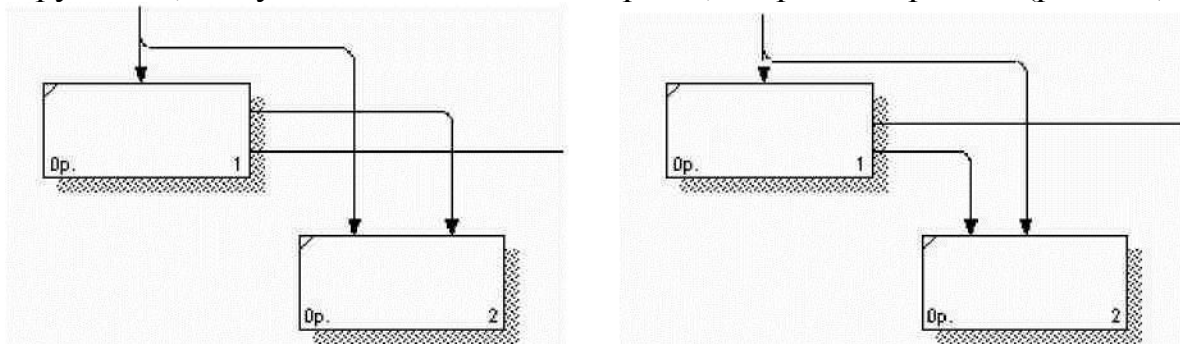


Рис. 5.2. Минимизация пересечений и поворотов стрелок

Если нужно изобразить связь по входу, то необходимо избегать нависания работ друг над другом. В этом случае ВРwin изображает связи по входу в виде петли, что затрудняет чтение диаграмм (рис. 5.3).

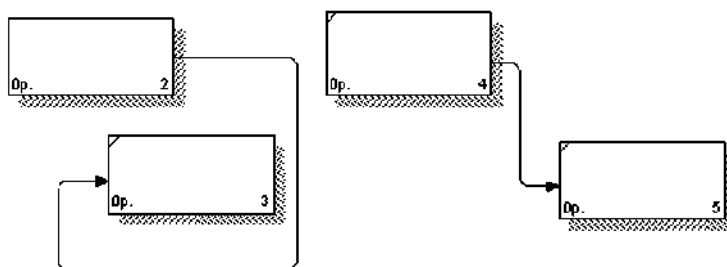


Рис. 5.3. Пример неправильного (слева) и правильного (справа) расположения работ при изображении связи по входу

Туннелирование стрелок. Вновь внесенные граничные стрелки на диаграмме декомпозиции нижнего уровня изображаются в квадратных скобках и автоматически не появляются на диаграмме верхнего уровня.

Если щелкнуть по кнопке *Resolve Border Arrow*, то стрелка мигрирует на диаграмму верхнего уровня, если по кнопке *Change To Tunnel*, то стрелка будет затуннелирована и не попадет на другую диаграмму. Тоннельная стрелка изображается с круглыми скобками на конце.

Туннелирование может быть применено для изображения малозначимых стрелок. Если на какой-либо диаграмме нижнего уровня необходимо изобразить малозначимые данные или объекты, которые не обрабатываются или не используются работами на текущем уровне, то их необходимо направить на вышестоящий уровень (на родительскую диаграмму). Если эти данные не используются на родительской диаграмме, то их нужно направить еще выше, и т. д. В результате малозначимая стрелка будет изображена на всех уровнях и затруднит чтение всех диаграмм, на которых она присутствует. Выходом является туннелирование стрелки на самом нижнем уровне. Такое туннелирование называется «не-в-родительской-диаграмме».

Другим примером туннелирование может быть ситуация, когда стрелка механизма мигрирует с верхнего уровня на нижний, причем на нижнем уровне этот механизм используется одинаково во всех работах без исключения. В этом случае стрелка механизма на нижнем уровне может быть удалена, после чего на родительской диаграмме она может быть затуннелирована, а в комментарии к стрелке или в словаре можно указать, что механизм будет использоваться во всех работах дочерней диаграммы декомпозиции. Такое туннелирование называется «не-в-дочерней-работе».

Слияние и расщепление моделей. Возможность слияния и расщепления моделей обеспечивает коллективную работу над проектом. Отдельная ветвь модели может быть отщеплена для использования в качестве независимой модели, для доработки или архивирования.

Brwin использует для слияния и разветвления моделей стрелки вызова. Для слияния необходимо выполнить следующие условия:

- обе сливаемые модели должны быть открыты в Brwin;
- имя модели-источника, которое присоединяют к модели-цели, должно совпадать с именем стрелки вызова работы в модели-цели;
- стрелка вызова должна исходить из недекомпозируемой работы (работа должна иметь диагональную черту в левом верхнем углу);
- имена контекстной работы подсоединяемой модели-источника и работы на модели-цели, к которой подсоединяют модель-источник, должны совпадать;
- модель-источник должна иметь по крайней мере одну диаграмму декомпозиции.

Для слияния моделей нужно щелкнуть правой кнопкой мыши по работе со стрелкой вызова в модели-цели и во всплывающем меню выбрать пункт *Merge Model*.

При слиянии моделей объединяются словари стрелок и работ. В случае одинаковых определений возможна перезапись определений или принятие

определений из модели-источника. То же относится к именам стрелок, хранилищам данных и внешним ссылкам.

После подтверждения слияния модель-источник подсоединяется к модели-цели, стрелка вызова исчезает, а работа, от которой отходила стрелка вызова, становится декомпозируемой – к ней подсоединяется диаграмма декомпозиции первого уровня модели-источника. Стрелки, касающиеся работы на диаграмме модели-цели, автоматически не мигрируют в декомпозицию, а отображаются как неразрешенные. Их следует туннелировать вручную.

В процессе слияния модель-источник остается неизменной и к модели-цели подключается фактически ее копия. Если в дальнейшем модель-источник будет редактироваться, то эти изменения автоматически не попадут в соответствующую ветвь модели-цели.

Разделение моделей производится аналогично. Для отщепления ветви от модели следует щелкнуть правой кнопкой мыши по декомпозированной работе и выбрать во всплывающем меню пункт *Split Model*. В появившемся диалоге *Split Options* следует указать имя создаваемой модели. После подтверждения расщепления в старой модели работа станет недекомпозированной, будет создана стрелка вызова, причем ее имя будет совпадать с именем новой модели, и, наконец, будет создана новая модель, причем имя контекстной работы будет совпадать с именем работы, от которой была "оторвана" декомпозиция.

Экспериментальная часть занятия

1. Переименуйте стрелку «Система оформления заказов» в стрелку «Бухгалтерская система».

2. Декомпозируем работу «Сборка и тестирование компьютеров» с четырьмя работами, используя методологию *IDEFO* по следующим критериям:

- производственный отдел получает заказы клиентов от отдела продаж по мере их поступления;
- диспетчер координирует работу сборщиков, сортирует заказы, группирует их и дает указание на отгрузку компьютеров, когда они готовы;
- каждые 2 ч диспетчер группирует заказы – отдельно для настольных компьютеров и ноутбуков – и направляет их на участок сборки;
- сотрудники участка сборки собирают компьютеры согласно спецификациям заказа и инструкциям по сборке. Когда группа компьютеров, соответствующая группе заказов, собрана, она направляется на тестирование. Тестеры проверяют каждый компьютер и в случае необходимости заменяют неисправные компоненты;
- тестеры направляют результаты тестирования диспетчеру, который на основании этой информации принимает решение о передаче компьютеров, соответствующих группе заказов, на отгрузку.

3. На основе перечисленных выше критериев внесите новые работы и стрелки (табл. 5.1 и 5.2).

Таблица 5.1

Работы диаграммы декомпозиции A2

<i>Activity Name</i>	<i>Activity Definition</i>
Отслеживание расписания и управление сборкой и тестированием	Просмотр заказов, установка расписания выполнения заказов, просмотр результатов тестирования, формирование групп заказов на сборку и отгрузку
Сборка настольных компьютеров	Сборка настольных компьютеров в соответствии с инструкциями и указаниями диспетчера
Сборка ноутбуков	Сборка ноутбуков в соответствии с инструкциями и указаниями диспетчера
Тестирование компьютеров	Тестирование компьютеров и компонентов. Замена неработающих компонентов

Таблица 5.2

Стрелки диаграммы декомпозиции A2

Имя стрелки	Источник	Тип источника	Назначение	Тип назначения
Диспетчер	Персонал производственного отдела	<i>Mechanism</i>	Отслеживание расписания и управление сборкой и тестированием	<i>Mechanism</i>
Заказы клиентов	Граница диаграммы	<i>Control</i>	Отслеживание расписания и управление сборкой и тестированием	<i>Control</i>
Заказы на настольные компьютеры	Отслеживание расписания и управление сборкой и тестированием	<i>Output</i>	Сборка настольных компьютеров	<i>Control</i>
Заказы на ноутбуки	Отслеживание расписания и управление сборкой и тестированием	<i>Output</i>	Сборка ноутбуков	<i>Control</i>
Компоненты	Туннелирование	<i>Input</i>	Сборка настольных компьютеров	<i>Input</i>
		<i>Input</i>	Сборка ноутбуков	<i>Input</i>
		<i>Input</i>	Тестирование компьютеров	<i>Input</i>
Настольные компьютеры	Сборка настольных компьютеров	<i>Output</i>	Тестирование компьютеров	<i>Input</i>

Стрелки диаграммы декомпозиции A2

Имя стрелки	Источник	Тип источника	Назначение	Тип назначения
Ноутбуки	Сборка ноутбуков	<i>Output</i>	Тестирование компьютеров	<i>Input</i>
Персонал производственного отдела	Туннелирование	<i>Mechanism</i>	Сборка настольных компьютеров	<i>Mechanism</i>
		<i>Mechanism</i>	Сборка ноутбуков	<i>Mechanism</i>
Правила сборки и тестирования	Граница диаграммы		Сборка настольных компьютеров	<i>Control</i>
			Сборка ноутбуков	<i>Control</i>
			Тестирование компьютеров	<i>Control</i>
Результаты сборки и тестирования	Сборка настольных компьютеров	<i>Output</i>	Граница диаграммы	<i>Output</i>
	Сборка ноутбуков	<i>Output</i>		
	Тестирование компьютеров	<i>Output</i>		
Результаты тестирования	Тестирование компьютеров	<i>Output</i>	Отслеживание расписания и управление сборкой и тестированием	<i>Input</i>
Собранные компьютеры	Тестирование компьютеров	<i>Output</i>	Граница диаграммы	<i>Output</i>
Тестер	Персонал производственного отдела	<i>Mechanism</i>	Тестирование компьютеров	<i>Mechanism</i>
Указания на отгрузку компьютеров	Отслеживание расписания и управление сборкой и тестированием	<i>Output</i>	Тестирование компьютеров	<i>Control</i>

5. Перейдите на диаграмму A0 и щелкните правой кнопкой мыши по работе «Отгрузка и получение». В контекстном меню выберите *Split Model*. В появившемся диалоге *Split Option* установите опцию *Enable Merge/Overwrite Option*, внесите имя новой модели – «Отгрузка и получение» и щелкните по кнопке *OK*. Обратите внимание, что у работы «Отгрузка и получение» появилась стрелка вызова. ВРwin создал также новую модель «Отгрузка и получение» (рис. 5.5).

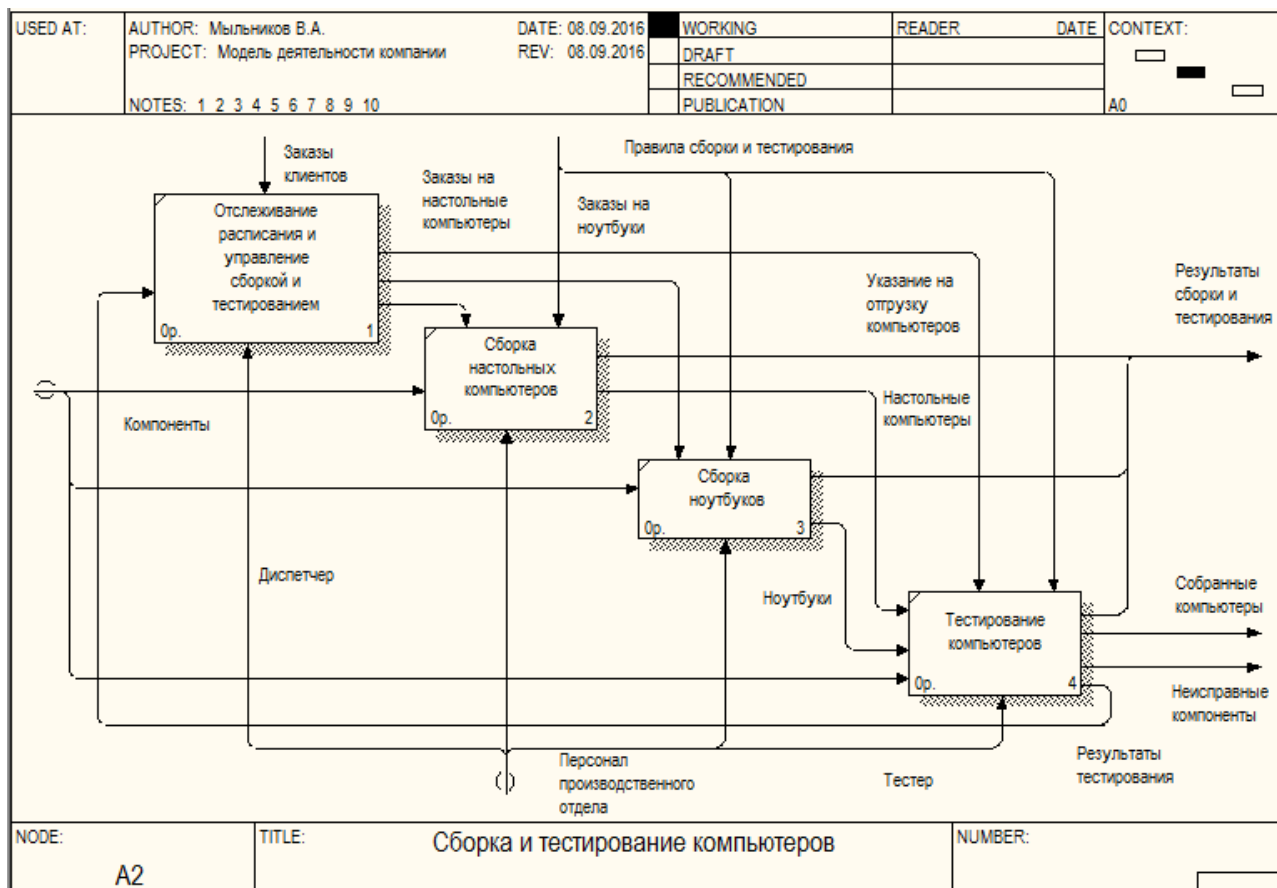


Рис. 5.4. Результат выполнения задания 4

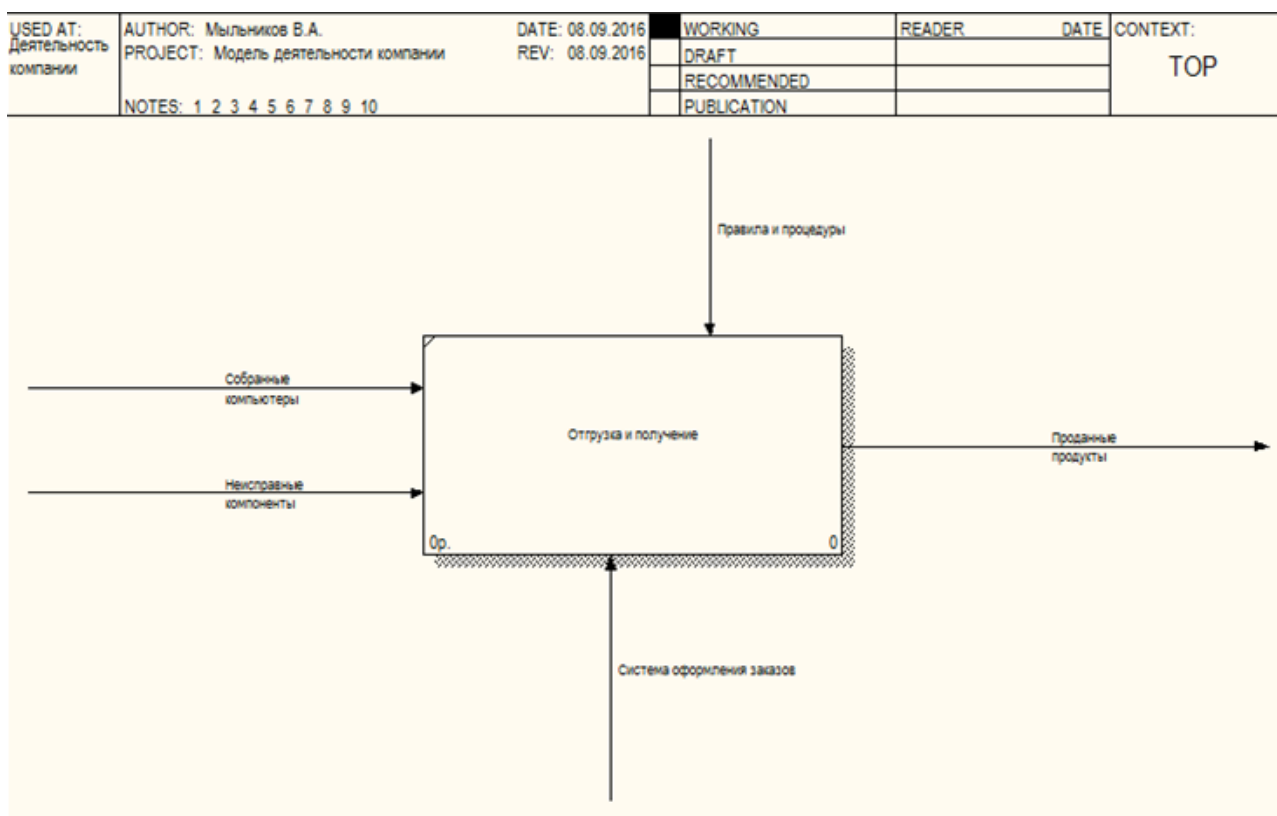


Рис. 5.5. Создание новой модели «Отгрузка и получение»

6. Внесите свойства новой модели (табл. 5.3).

Таблица 5.3

Свойства новой модели

Наименование параметра	Значение параметра
<i>Time Frame</i>	AS-IS
<i>Purpose</i>	Документировать работу «Отгрузка и получение»
<i>Viewpoint</i>	Начальник отдела
<i>Definition</i>	Модель создается для иллюстрации возможности по расщеплению и слиянию моделей
<i>Scope</i>	Работы по получению комплектующих и отправке готовой продукции

7. Сохраните новую модель.

8. Декомпозируйте контекстную работу на три работы по методологии *IDEF0* (табл. 5.4).

Таблица 5.4

Декомпозиция работы «Отгрузка и получение»

<i>Activity Name</i>	<i>Activity Definition</i>
Получить комплектующие	Физически получить комплектующие и сделать соответствующие записи в информационной системе
Доставить комплектующие	Доставить комплектующие сборщикам и тестерам
Отгрузить товар и возврат	Отгрузить товар клиентам и неисправные компоненты (возврат) поставщикам

9. Свяжите граничные стрелки, как показано на рис. 5.6.

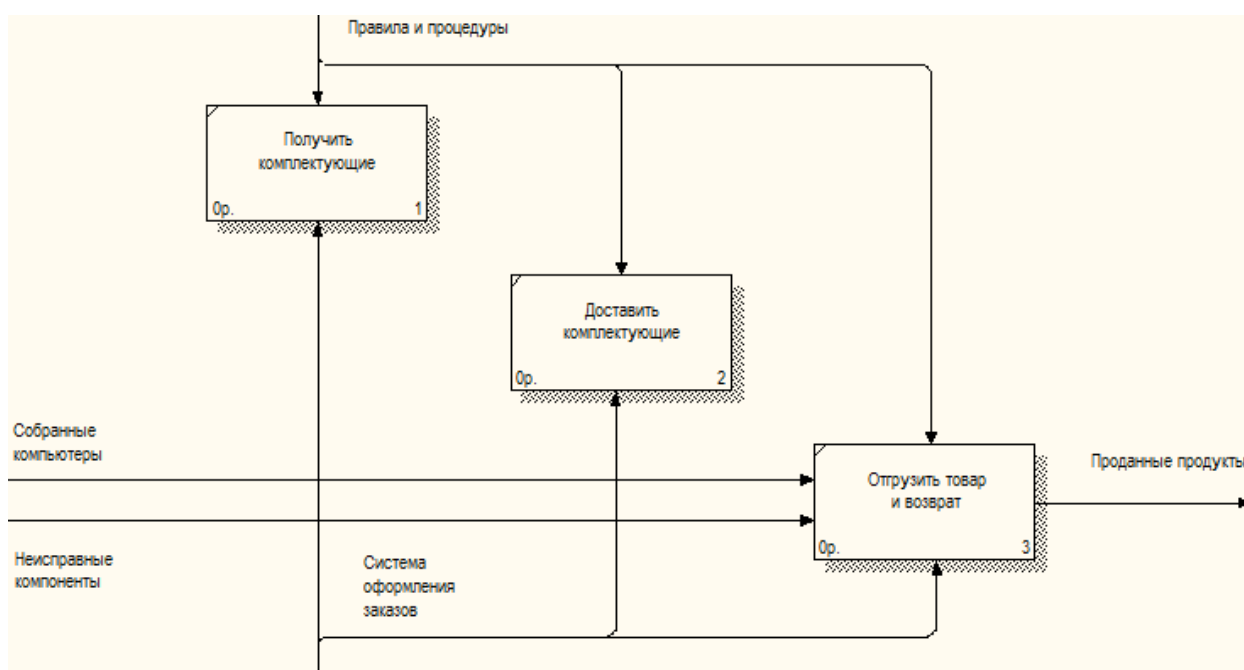


Рис. 5.6. Внутренние стрелки на диаграмме «Отгрузка и получение»

10. Внесите внутренние и граничные стрелки (табл. 5.5).

Таблица 5.5

**Внутренние и граничные стрелки на декомпозиции работы
«Отгрузка и получение»**

<i>Arrow Name</i>	<i>Arrow Definition</i>
Возврат поставщику	Неисправные компоненты
Компоненты	Выберите название из списка (словаря)
Компоненты от поставщика	Компоненты от поставщика
Проверенные компоненты	Проверенные и подготовленные для передачи сборщикам и тестировщикам компоненты

11. Результат добавления стрелок показан на рис. 5.7.

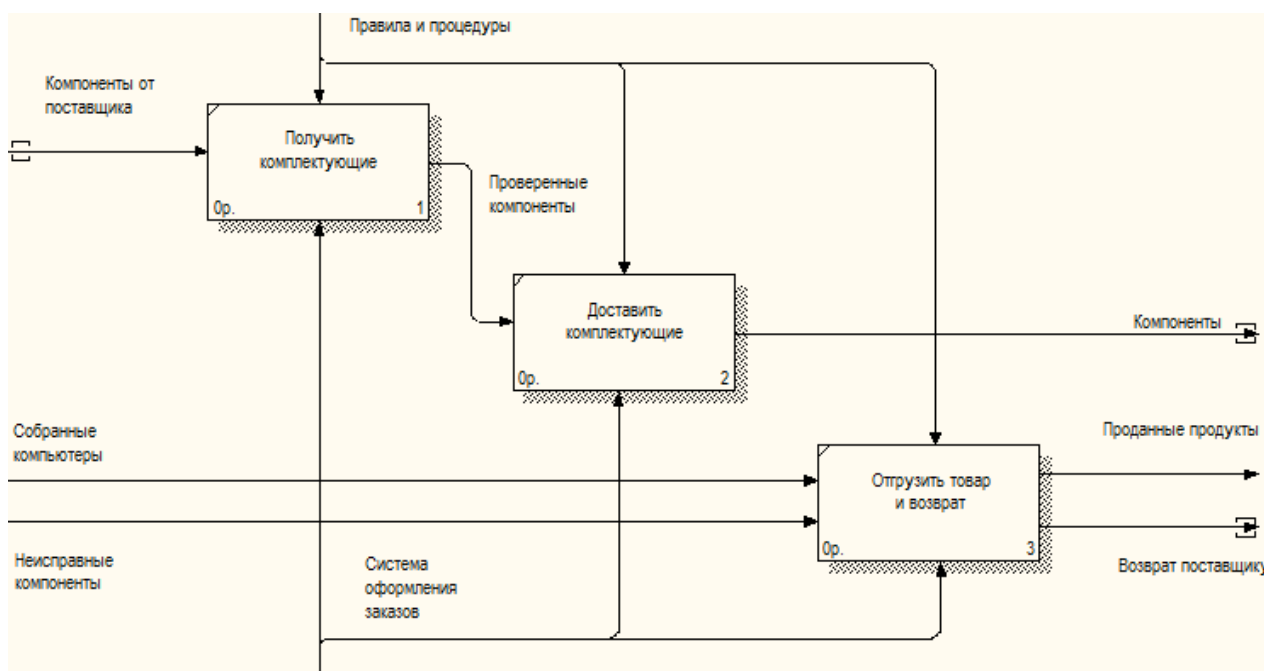


Рис. 5.7. Внутренние и граничные стрелки на диаграмме «Отгрузка и получение»

12. Туннелируйте граничные стрелки (Arrow Tunnel/Resolve Border Arrow). Диаграмма декомпозиции показана на рис. 5.8. Родительская диаграмма «Отгрузка и получение» изображена на рис. 5.9.

13. Перейдите в модель «Деятельность компании». На диаграмме АО щелкните правой кнопкой мыши по работе «Отгрузка и получение». В контекстном меню выберите *Merge Model*. В появившемся диалоге *Merge Model* установите опцию *Cut/Paste entire dictionaries* и щелкните по *OK*.

Обратите внимание, что у работы «Отгрузка и получение» исчезла стрелка вызова и появилась новая декомпозиция.

14. На диаграмме АО туннелируйте (Arrow Tunnel/Resolve Border Arrow) и свяжите стрелки, как показано на рис. 5.10.

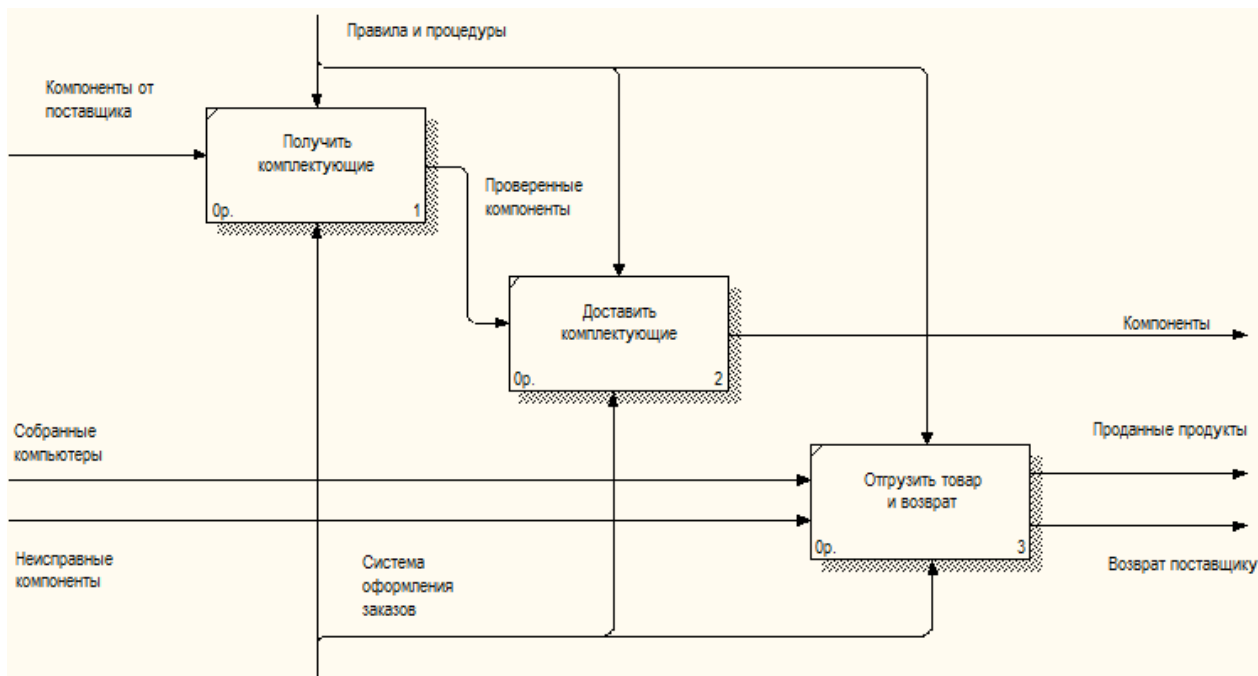


Рис. 5.8. Диаграмма декомпозиции работы «Отгрузка и получение»

15. В связи с появлением стрелки входа «Компоненты» у работы «Сборка и тестирование компьютеров» удалите несвязанную стрелку «Компоненты» и ее ветви на диаграмме А2 «Сборка и тестирования компьютеров» и соответственно продолжите стрелку П1 «Компоненты». Проверьте, является ли стрелка «Компоненты» стрелкой выхода (О) на диаграмме А3 «Отгрузка и получение» (рис. 5.11).

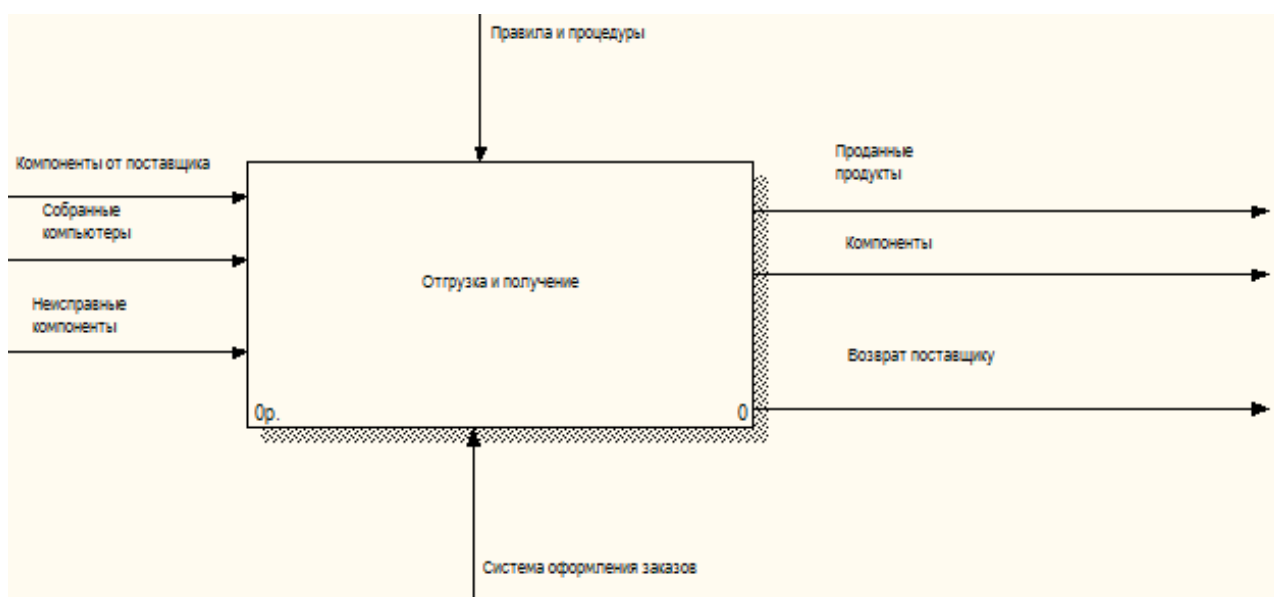


Рис. 5.9. Родительская диаграмма «Отгрузка и получение»

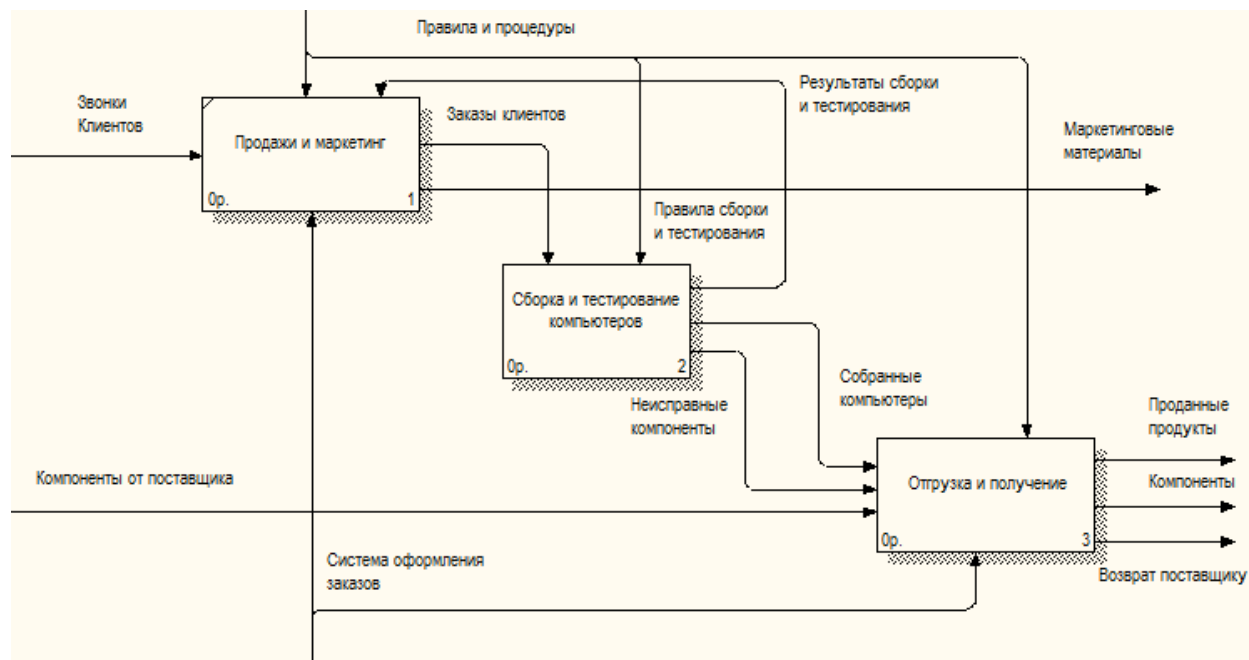


Рис. 5.10. Диаграмма «Деятельность компании» после слияния моделей

16. Создайте новую модель «ТЕСТ».

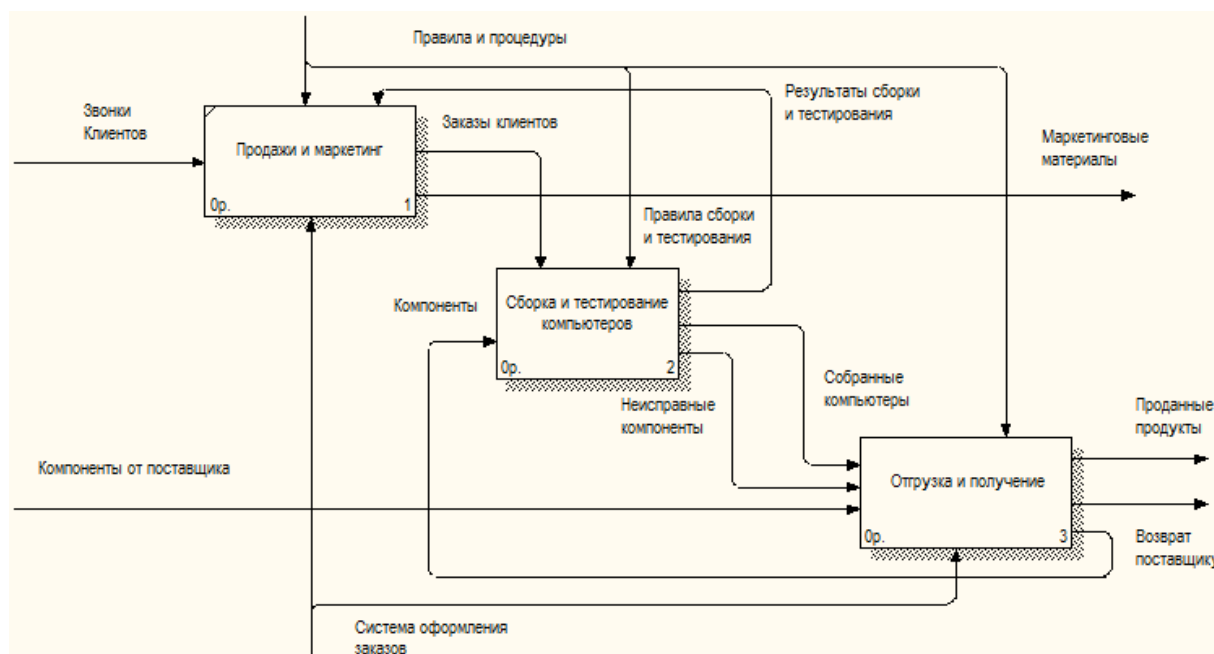


Рис. 5.11. Диаграмма «Деятельность компании» после изменения стрелки «Компоненты»

17. Декомпозируйте контекстную работу в новой модели, но не вносите имена работ. Переключите *Model Explorer* во вкладку *Activity*.

18. С помощью *drag&drop* перенесите любую работу из модели «Деятельность компании» на диаграмму декомпозиции модели «ТЕСТ». Для этого выделите работу для копирования, нажав *ctrl* и удерживая левую клавишу мыши на пиктограмме работы, перенесите работу в нужное место на диаграмме.

19. В диалоговом окне «*Continue with Merge?*» установите опцию *Cut/Paste entire dictionaries* и щелкните по кнопке *OK*.

20. В результате работа из модели «Деятельность компании» копируется на новую диаграмму модели «ТЕСТ».

21. Щелкните по работе в модели «ТЕСТ» и переместите работу на нужное место на другой диаграмме.

22. В появившемся диалоге *Continue with Merge?* щелкните *OK*. В результате работа переносится из одной диаграммы на другую.

Контрольные вопросы

1. Что такое туннелирование?
2. Типы туннелирования и их отличия.
3. Как правильно называются стрелки при слиянии и расщеплении модели?
4. Что такое *Squiggle*?
5. Как произвести слияние и расщепление моделей?
6. Каким образом можно скопировать работу?
7. Какое имя надо задать для новой модели при расщеплении?
8. Что означает опция *Cut/Paste entire dictionaries*?
9. Какие условия необходимо выполнить для слияния моделей?

Практическое занятие № 6

ТЕМА: «МЕТОДОЛОГИЯ ОПИСАНИЯ ПРОЦЕССОВ IDEF3»

Цель занятия: изучить методологии функционального моделирования IDEF3.

Задача: выполнить декомпозицию модели IDEF3.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Выполнение декомпозиции модели IDEF3 для описания пошаговых операций вычислительных процессов до потоков данных.
3. Установление связи с внешними системами и хранилищами модели.

Теоретические основы занятия

Методология построения моделей *IDEF3*, называемая также *Workflow diagramming* – это методология моделирования, которая необходима для описания логики взаимодействия информационных потоков. Данная методология использует графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов. Диаграммы *Workflow* могут быть применены в моделировании бизнес-процессов для анализа завершенности процедур обработки информации. С их помощью можно описывать сценарии действий сотрудников организации, например последовательность обработки заказа или события, которые необходимо обработать за конечное время. Каждый сценарий сопровождается описанием процесса и может быть использован для документирования каждой функции.

IDEF3 – это метод, имеющий основной целью дать возможность аналитикам описать ситуацию, когда процессы выполняются в определенной последовательности, а также описать объекты, участвующие совместно в одном процессе. Техника описания набора данных *IDEF3* является частью структурного анализа. В отличие от некоторых методик описаний процессов *IDEF3* не ограничивает аналитика чрезмерно жесткими рамками синтаксиса, что может привести к созданию неполных или противоречивых моделей. *IDEF3* может быть также использован как метод создания процессов. Каждая работа в *IDEF3* описывает какой-либо сценарий бизнес-процесса и может являться составляющей другой работы. Поскольку сценарий описывает цель и рамки модели, важно, чтобы работы именовались отглагольным существительным, обозначающим процесс действия, или фразой, содержащей такое существительное.

Точка зрения на модель, цель модели – те вопросы, на которые призвана ответить модель – должны быть задокументированы.

Диаграммы. Диаграмма является основной единицей описания в *IDEF3*. Важно правильно построить диаграммы, поскольку они предназначены для чтения другими людьми (а не только автором).

Единицы работы. Это – *Unit of Work (UOW)*, также называемые работами, являются центральными компонентами модели. В *IDEF3* работы изображаются прямоугольниками с прямыми углами и имеют имя, выраженное отглагольным существительным, обозначающим процесс действия, одиночным или в составе фразы, и номер (идентификатор); другое имя существительное в составе той же фразы обычно отображает основной выход (результат) работы (например, «Изготовление изделия»). Часто имя существительное в имени работы меняется в процессе моделирования, поскольку модель может уточняться и редактироваться. Идентификатор работы присваивается при создании и не меняется никогда. Даже если работа будет удалена, ее идентификатор не будет вновь использоваться для других работ. Обычно номер работы состоит из номера родительской работы и порядкового номера на текущей диаграмме.

Связи. Связи показывают взаимоотношения работ. Все связи в *IDEF3* однонаправленные и могут быть направлены куда угодно, но обычно диаграммы *IDEF3* стараются построить так, чтобы связи были направлены слева направо. В *IDEF3* различают три типа стрелок, изображающих связи, стиль которых устанавливается через меню *Arrow Properties*. Рассмотрим их.

Старшая связь (Precedence) – это сплошная линия, связывающая единицы работ (*UOW*). Рисуются слева направо или сверху вниз. Показывает, что работа-источник должна закончиться прежде, чем начнется работа-цель.

Отношения (Relational Link) – это пунктирная линия, используемая для изображения связей между единицами работ (*UOW*) а также между единицами работ и объектами ссылок.






Потоки объектов (Object Flow) – это стрелка с двумя наконечниками, применяется для описания того факта, что объект используется в двух или более единицах работы, например, когда объект порождается в одной работе и используется в другой.

Старшая связь и поток объектов. Это отношение показывает, что стрелка является альтернативой старшей стрелке или потоку объектов в смысле задания последовательности выполнения работ – работа-источник не обязательно должна закончиться, прежде чем работа-цель начнется. Работа-цель может закончиться прежде, чем закончится работа-источник.

Перекрестки (Junction). Окончание одной работы может служить сигналом к началу нескольких работ или же одна работа для своего запуска может ожидать окончания нескольких работ. Перекрестки используются для отображения логики взаимодействия стрелок при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы. Различают перекрестки для слияния (*Fan-in Junction*) и разветвления (*Fan-out Junction*) стрелок. Перекресток не может использоваться одновременно для слияния и для разветвления. Смысл каждого типа перекрестков приведен в табл. 6.1.

Все перекрестки на диаграмме нумеруются, каждый номер имеет префикс *J*. Можно редактировать свойства перекрестка при помощи диалога *Definition Editor*. В отличие от *IDEF0* и *DFD* в *IDEF3* стрелки могут сливаться и разветвляться только через перекрестки.

Типы перекрестков

Соединение	Имя	Значение <i>Fan-in</i>	Значение <i>Fan-out</i>
	Асинхронное <i>AND</i>	Все последующие процессы должны быть полными	Все предшествующие процессы должны быть полными
	Синхронное <i>AND</i>	Все следующие процессы обрабатываются одновременно комплексно	Все предшествующие процессы начинают обрабатываться одновременно
	Асинхронное <i>OR</i>	Один или более предшествующих процессов должны быть завершены	Один или более последующих процессов должны быть начаты
	Синхронное <i>OR</i>	Один или более предшествующим процессам завершаются одновременно	Один или более последующих процессов начинаются одновременно
	<i>XOR</i>	Ровно один предшествует завершающемуся процессу	Ровно один следует за началом процесса

Объект ссылки. Объект ссылки в *IDEF3* выражает некую идею, концепцию или данные, которые нельзя связать со стрелкой, перекрестком или работой. Объект ссылки изображается в виде прямоугольника, похожего на прямоугольник работы. Имя объекта ссылки задается в диалоге *Referent* (пункт всплывающего меню *Name Editor*), в качестве имени можно использовать имя какой-либо стрелки с других диаграмм или имя сущности из модели данных. Объекты ссылки должны быть связаны с единицами работ или перекрестками пунктирными линиями. Официальная спецификация *IDEF3* различает три стиля объектов ссылок – безусловные (*unconditional*), синхронные (*synchronous*) и асинхронные (*asynchronous*). ВРwin поддерживает только безусловные объекты ссылок. Синхронные и асинхронные объекты ссылок, используемые в диаграммах переходов состояний объектов, не поддерживаются.

При внесении объектов ссылок помимо имени следует указывать тип объекта ссылки. Типы объектов ссылок приведены в табл. 6.2.

Декомпозиция работ. В *IDEF3* декомпозиция используется для детализации работ. Методология *IDEF3* позволяет декомпозировать работу многократно, т.е. работа может иметь множество дочерних работ. Это позволяет в одной модели описать альтернативные потоки. Возможность множественной декомпозиции предъявляет дополнительные требования к нумерации работ. Так, номер работы состоит из номера родительской работы, версии декомпозиции и собственного номера работы на текущей диаграмме.

Рассмотрим процесс декомпозиции диаграмм *IDEF3*, включающий взаимодействие автора (аналитика) и одного или нескольких экспертов предметной области.

Типы объектов ссылок

Тип объекта ссылки	Цель описания
<i>OBJECT</i>	Описывает участие важного объекта в работе
<i>GOTO</i>	Инструмент циклического перехода (в повторяющейся последовательности работ), возможно на текущей диаграмме, но не обязательно. Если все работы цикла присутствуют на текущей диаграмме, цикл может также изображаться стрелкой, возвращающейся на стартовую работу: <i>GOTO</i> может ссылаться на перекресток
<i>UOB</i> (Unit of behavior)	Применяется, когда необходимо подчеркнуть множественное использование какой-либо работы, но без цикла. Например, работа «Контроль качества» может быть использована в процессе «Изготовления изделия» несколько раз, после каждой единичной операции. Обычно этот тип ссылки не используется для моделирования автоматически напускающихся работ
<i>NOTE</i>	Используется для документирования важной информации, относящейся к каким-либо графическим объектам на диаграмме. <i>NOTE</i> является альтернативой внесению текстового объекта в диаграмму
<i>ELAB</i> (Elaboration)	Используется для усовершенствования графиков или их более детального описания. Обычно употребляется для детального описания разветвления и слияния стрелок на перекрестках

Описание сценария, области и точки зрения. Перед проведением сеанса экспертизы у экспертов предметной области должны быть задокументированы сценарии и рамки модели для того, чтобы эксперт мог понять цели декомпозиции. Если точка зрения моделирования отличается от точки зрения эксперта, то она должна быть особенно тщательно задокументирована.

Возможно, что эксперт самостоятельно не сможет передать необходимую информацию. В этом случае аналитик должен приготовить список вопросов для проведения интервью.

Определение работ и объектов. Обычно эксперт предметной области передает аналитику текстовое описание сценария. В дополнение к этому может существовать документация, описывающая представляющие интерес процессы. Из всей этой информации аналитик должен составить список кандидатов на работы (отглагольные существительные, обозначающие процесс, одиночные или в составе фразы) и кандидатов на объекты (существительные, обозначающие результат выполнения работы), которые необходимы для перечисленных в списке работ.

В некоторых случаях целесообразно создать графическую модель для представления ее эксперту предметной области. Графическая модель может быть также создана после сеанса сбора информации для того, чтобы детали форматирования диаграммы не смущали участников.

Поскольку разные фрагменты модели *IDEF3* могут быть созданы разными группами аналитиков в разное время, *IDEF3* поддерживает простую схему

нумерации работ в рамках всей модели. Разные аналитики оперируют разными диапазонами номеров, работая при этом независимо.

Последовательность и согласование. Если диаграмма создается после проведения интервью, аналитик должен принять некоторые решения, относящиеся к иерархии диаграмм, например, сколько деталей включать в одну диаграмму. Если последовательность и согласование диаграмм неочевидны, может быть проведена еще одна экспертиза для детализации и уточнения информации. Важно различать подразумевающее согласование (согласование, которое подразумевается в отсутствие связей) и ясное согласование (согласование, ясно изложенное в мнении эксперта).

Работы, перекрестки и документирование объектов. *IDEF3* позволяет внести информацию в модель различными способами. Например, логика взаимодействия может быть отображена графически в виде комбинации перекрестков. Та же информация может быть отображена в виде объекта ссылки типа *ELAB* (*Elaboration*). Это позволяет аналитику вносить информацию в удобном в данный момент времени виде. Важно учитывать, что модели могут быть реорганизованы, например, для их представления в более презентабельном виде. Выбор формата для презентации часто имеет важное значение для организации модели, поскольку комбинация перекрестков занимает значительное место на диаграмме и использование иерархии перекрестков затрудняет расположение работ на диаграмме.

В результате дополнения диаграмм *IDEF0* диаграммами *IDEF3* может быть создана смешанная модель, которая наилучшим образом описывает все стороны деятельности предприятия. Иерархию работ в смешанной модели можно увидеть в окне *Model Explorer*. Работы в нотации *IDEF0* изображаются зеленым цветом, *IDEF3* – желтым.

Экспериментальная часть занятия

1. Перейдите на диаграмму *A2* и декомпозируйте работу «Сборка настольных компьютеров». В диалоге *Activity Box Count* установите число работ, равное единице, и нотацию *IDEF3*.

Возникает диаграмма *IDEF3*, содержащая работы (*UOW*). Правой кнопкой мыши щелкните по работе, выберите в контекстном меню *Name* и внесите имя работы «Подготовка компонентов». Затем во вкладке *Definition* внесите определение «Подготавливаются все компоненты компьютера согласно спецификации заказа».


2. Во вкладке *UOW* внесите свойства работы (табл. 6.3).

Таблица 6.3


Свойства *UOW*

<i>Objects</i>	Компоненты: винчестеры, корпуса, материнские платы, видеокарты, дисковод <i>DVD-RW</i> , card-reader, программное обеспечение
<i>Facts</i>	Доступные операционные системы: <i>Windows 7</i> , <i>Windows 8</i> , <i>Windows 10</i>
<i>Constrains</i>	Установка модема требует установки дополнительного программного обеспечения

3. Внесите в диаграмму еще пять работ: «Установка материнской платы и винчестера», «Установка DVD-RW», «Установка card-reader», «Инсталляция операционной системы», «Инсталляция дополнительного программного обеспечения».

4. С помощью кнопки  палитры инструментов создайте объект ссылки. Внесите имя объекта внешней ссылки «Компоненты» (выбрать из существующих стрелок). Свяжите стрелкой объект ссылки и работу «Подготовка компонентов» (стиль стрелки: ссылка - *Referent*).

5. Свяжите стрелкой работы «Подготовка компонентов» (выход) и «Установка материнской платы и винчестера» (вход). Измените стиль стрелки на *Object Flow*.

6. С помощью кнопки  на палитре инструментов внесите два перекрестка типа «асинхронное ИЛИ» и свяжите работы с перекрестками, как показано на рис. 6.1.

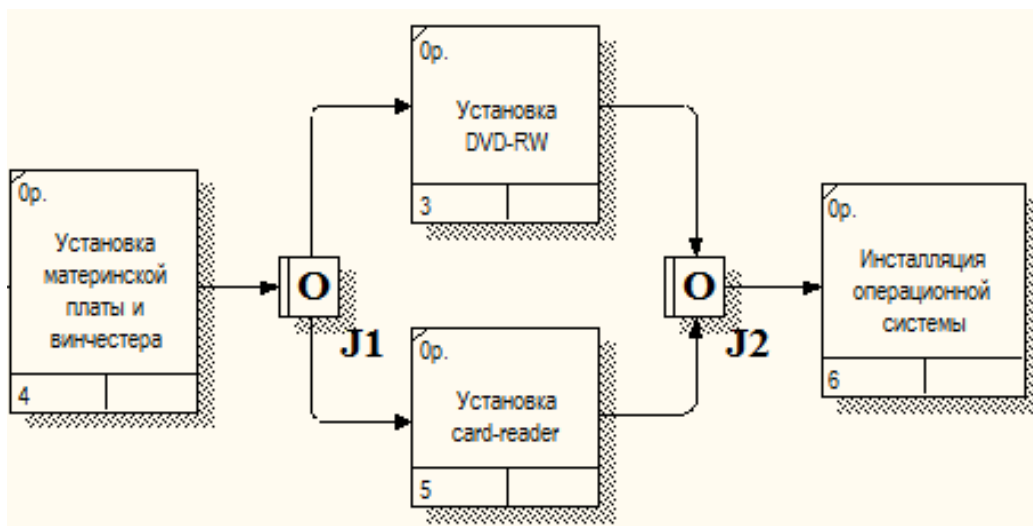


Рис. 6.1. Создание перекрестков на диаграмме

Правой кнопкой щелкните по перекрестку для разветвления (*fan-out*), выберите *Name* и внесите имя «Компоненты, требуемые в спецификации заказа».

7. Создайте два перекрестка типа «исключающего ИЛИ» и свяжите работы; добавьте объект внешней ссылки и дайте ему имя «Программное обеспечение». Для стрелки, соединяющей «Компоненты» и работу «Подготовка компонентов», измените тип стрелы на *Referent*. Для стрелки, соединяющей работу «Инсталляция дополнительного программного обеспечения» и «Программное обеспечение», измените тип стрелы на *Referent*. Для стрелки, соединяющей работу «Инсталляция операционной системы» и «Программное обеспечение», измените тип стрелы на *Referent*. Результат данной лабораторной работы должен быть, как на рис. 6.2.

8. Выберите пункт меню *Diagram/Add IDEF3 Scenario*. Создайте диаграмму сценария (декомпозиция диаграммы «Сборка настольных компьютеров») на основе диаграммы *IDEF3* «Сборка настольных компьютеров» (A22). Присвойте ей имя «Мой сценарий». Отметьте галочкой *Copy contents of source diagram*.

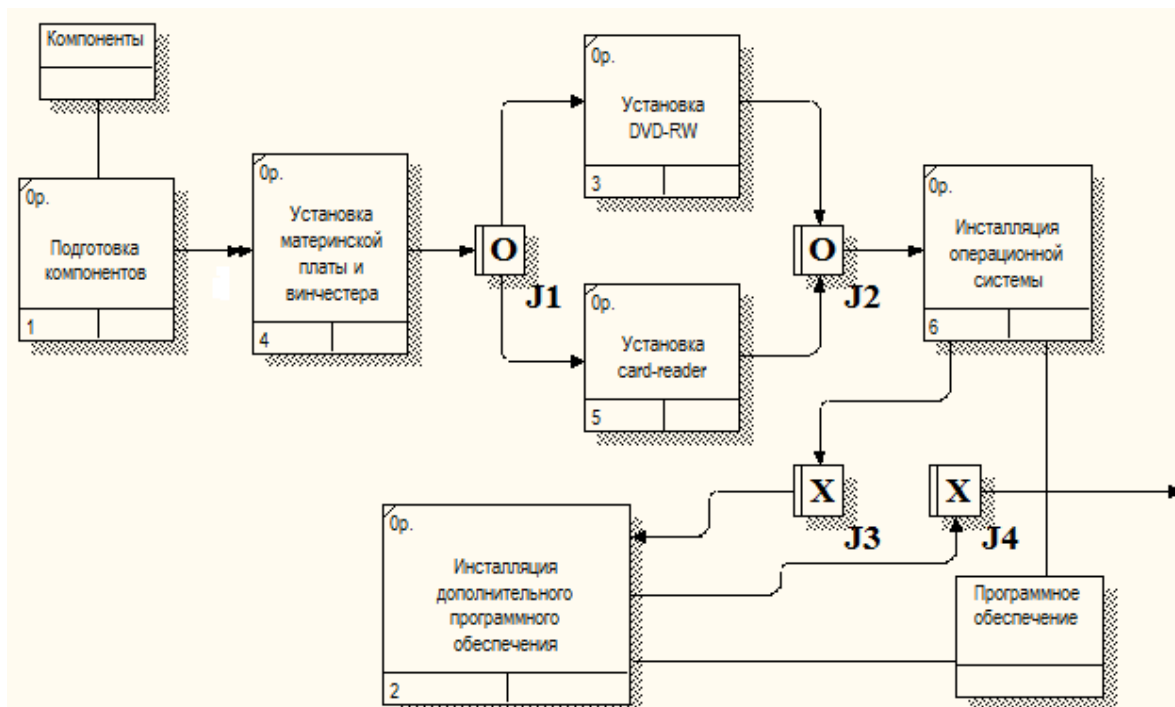


Рис. 6.2. Результат выполнения упражнения 7

9. Удалите элементы, не входящие в сценарий, как показано на рис. 6.3.

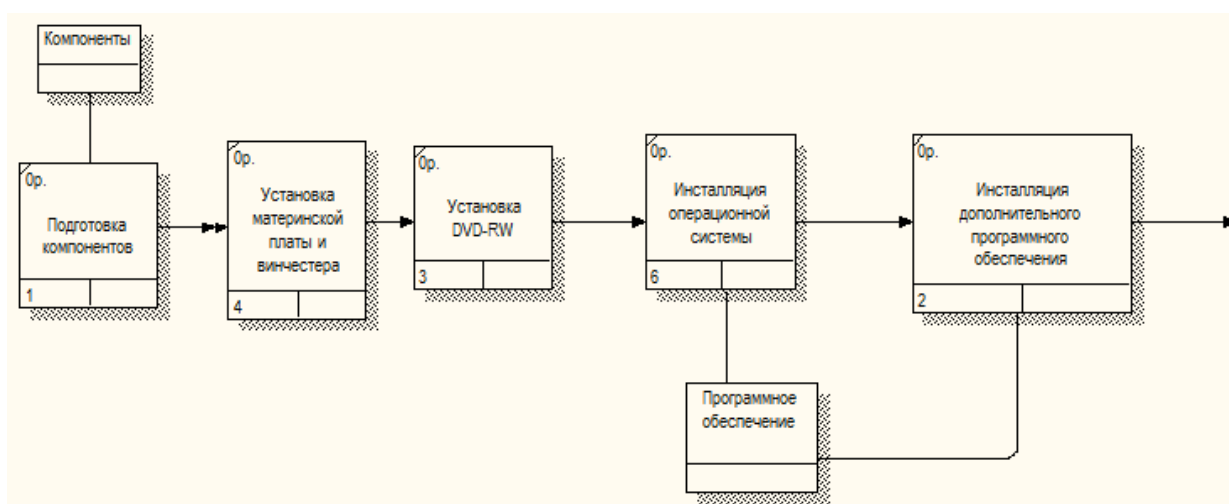


Рис. 6.3. Результат выполнения работы

Контрольные вопросы

1. Чем отличаются диаграммы *IDEF3* от диаграмм *IDEF0*?
2. Какие бывают виды перекрестков?
3. Что такое объект ссылки?
4. Для чего необходимо строить *IDEF3*-сценарий?
5. Что такое имитационное моделирование?

Практическое занятие № 7

ТЕМА: «СТОИМОСТНЫЙ АНАЛИЗ И СВОЙСТВА, ОПРЕДЕЛЯЕМЫЕ ПОЛЬЗОВАТЕЛЕМ»

Цель занятия: изучить методологию функционально-стоимостного анализа и управления дополнительными свойствами объектов.

Задачи:

1. Изучить методологию функционально-стоимостного анализа.
2. Изучить методику управления дополнительных свойств объектов.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Выполнение функционально-стоимостного анализа и первоначальная оценка разрабатываемой системы.
3. Создание структуры дополнительных свойств объектов модели.

Теоретические основы занятия

ВРwin предоставляет аналитику два инструмента для оценки модели – стоимостный анализ, основанный на работах (*Activity Based Costing, ABC*), и свойства, определяемые пользователем (*User Defined Properties, UDP*).

ABC является широко распространенной методикой, используемой международными корпорациями и государственными организациями для идентификации истинных движителей затрат в организации.

Стоимостный анализ представляет собой соглашение об учете, используемое для сбора затрат, связанных с работами, с целью определить общую стоимость процесса. Стоимостный анализ основан на модели работ, потому что количественная оценка невозможна без детального понимания функциональности предприятия. Обычно *ABC* применяется для того, чтобы понять происхождение выходных затрат и облегчить выбор нужной модели работ при реорганизации деятельности предприятия (*Business Process Reengineering, BPR*). С помощью стоимостного анализа можно решить такие задачи, как определение действительной стоимости производства продукта, определение действительной стоимости поддержки клиента, идентификация работ, которые стоят больше всего (те, которые должны быть улучшены в первую очередь), обеспечение менеджеров финансовой мерой предлагаемых изменений т.д.

ABC может проводиться только тогда, когда модель работы последовательная (следует синтаксическим правилам *IDEF0*), корректная (отражает бизнес), полная (охватывает всю рассматриваемую область) и стабильная (проходит цикл экспертизы без изменений), другими словами, когда создание модели работы закончено.

ABC включает ряд основных понятий. Ими являются:

- объект затрат – причина, по которой работа выполняется, обычно, основной выход работы, стоимость работ есть суммарная стоимость объектов затрат;

- движитель затрат — характеристики входов и управлений, которые влияют на то, как выполняется и как долго длится работа;
- центры затрат, которые можно трактовать как статьи расхода.

При проведении стоимостного анализа сначала задаются единицы измерения времени и денег. Если в списке выбора отсутствует необходимая валюта (например, рубль), то ее можно добавить. Символ валюты по умолчанию берется из настроек *Windows*. Диапазон измерения времени в списке *Unit of measurement* достаточен для большинства случаев: от секунд до лет.

Затем описываются центры затрат (*Cost centers*). Для внесения центров затрат необходимо вызвать диалог *Cost Center Editor* (меню *Edit/ABC Cost Centers*). Каждому центру затрат следует дать подробное описание в окне *Definition*. Список центров затрат упорядочен. Порядок в списке можно менять при помощи стрелок, расположенных справа от списка. Задание определенной последовательности центров затрат в списке облегчает последующую работу при присвоении стоимости работам и имеет значение при использовании единых стандартных отчетов в разных моделях. Информация о центрах затрат и *UDP* сохраняется в виде указателей, т.е. хранятся номера центров затрат. Поэтому, если нужно использовать один и тот же стандартный отчет в разных моделях, списки центров затрат должны быть в них одинаковы.

Общие затраты по работе рассчитываются как сумма по всем центрам затрат. При вычислении затрат вышестоящей (родительской) работы сначала вычисляется произведение затрат дочерней работы на частоту работы (число раз, которое работа выполняется в рамках проведения родительской работы), затем результаты складываются. Если во всех работах модели включен режим *Compute from Decompositions*, то подобные вычисления автоматически проводятся по всей иерархии работ снизу вверх. Этот принцип подсчета справедлив, если работы выполняются последовательно.

Встроенные возможности *BPwin* позволяют разрабатывать упрощенные модели стоимости, которые оказываются полезными при предварительной оценке затрат. Если схема выполнения более сложная (например, работы производятся альтернативно), то можно отказаться от подсчета и задать итоговые суммы для каждой работы вручную (*Override Decompositions*). В этом случае результаты расчетов с нижних уровней декомпозиции будут игнорироваться, при расчетах на верхних уровнях будет учитываться сумма, заданная вручную. На любом уровне результаты расчетов сохраняются независимо от выбранного режима, поэтому при выключении опции *Override Decompositions* расчет снизу вверх производится обычным образом.

Результаты стоимостного анализа могут существенно повлиять на очередность выполнения работ.

Результаты стоимостного анализа наглядно представляются в специальном отчете *BPwin* — *Activity Cost Report* (меню *Report/Activity Cost Report*). Отчет позволяет документировать имя, номер, определение и стоимость работ как суммарную, так и отдельно по центрам затрат. Результаты отображаются и непосредственно на диаграммах. В левом нижнем углу прямоугольника работы может показываться либо стоимость (по умолчанию), либо продолжительность,

либо частота проведения работы. Настройка отображения осуществляется в диалоге *Model Properties* (меню *Edit/Model Properties*), закладка *Display*, *ABC Data*, *ABC Units*.

ABC позволяет оценить стоимостные и временные характеристики системы. Если стоимостных показателей недостаточно, то имеется возможность внесения собственных метрик – свойств, определенных пользователем (*User Defined Properties*, *UDP*). *UDP* позволяют провести дополнительный анализ, хотя и без суммирующих подсчетов.

Для описания *UDP* служит диалог *User-Defined Property Name Editor* (меню *Edit/UDP Definition*). В верхнем окне диалога вносится имя *UDP*, в списке выбора *Datatype* описывается тип свойства. Имеется возможность задания 18 различных типов *UDP*, в том числе управляющих команд и массивов, объединенных по категориям. Для внесения категории следует задать имя категории в окне *New Category/Member* и щелкнуть по кнопке *Add Category*. Для присвоения свойства категории необходимо выбрать *UDP* из списка, затем категорию из списка категорий и щелкнуть по кнопке *Update*. Одна категория может объединять несколько свойств, в то же время одно свойство может входить в несколько категорий. Свойство типа *List* может содержать массив предварительно определенных значений. Для определения области значений *UDP* типа *List* следует задать значение свойства в окне *New Category/Member* и щелкнуть по кнопке *Add Member*. Значения из списка можно редактировать и удалять.

С каждой работой можно поставить в соответствие набор *UDP*. Для этого следует на работе щелкнуть правой клавишей мыши и в контекстном меню выбрать пункт *UDP Editor*. В закладке *UDP Values* диалога *IDEF0 Activity Properties* можно задать значения *UDP*. Свойства типа *List* отображаются списком выбора, который заполнен предварительно определенными значениями. Свойства типа *Command* могут иметь в качестве значения командную строку, которая выполняется при нажатии на кнопку.

Кнопка *Categories* служит для задания фильтра по категориям *UDP*. По умолчанию в списке показываются свойства всех категорий,

В левом нижнем углу диалога настройки отчета показывается список *UDP*. С помощью кнопки *Activity Categories* можно установить фильтр по категориям.

Экспериментальная часть занятия

1. В диалоге *Model Properties* (вызывается из меню *Model/Model Properties*) во вкладке *ABC Units* (рис. 7.1) установите единицы измерения и денег и времени – рубли и дни.

2. Перейдите в *Dictionary/Cost Center* и в диалоге *Cost Center Dictionary* внесите название и определение центров затрат (табл. 7.1). Для подтверждения ввода данных используйте клавишу табуляции.

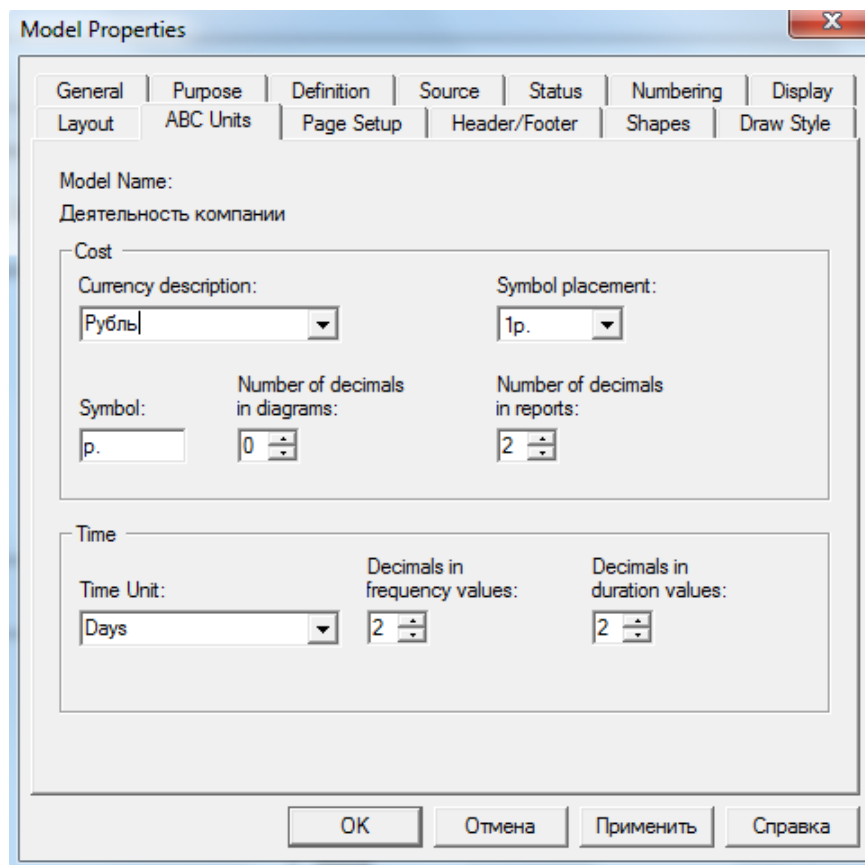


Рис. 7.1. Вкладка *ABC Units* диалога *Model Properties*

Таблица 7.1

Цены затрат ABC

Центр затрат	Определение
Управление	Затраты на управление, связанные с составлением графика работ, формированием партий компьютеров, контролем над сборкой и тестированием
Рабочая сила	Затраты на оплату рабочих, занятых сборкой и тестированием компьютеров
Компоненты	Затраты на закупку компонентов

3. Для отображения стоимости каждой работы в нижнем левом углу прямоугольника перейдите в меню *Model/Model Properties* и во вкладке *Display* диалога *Model Properties* включите опцию *ABC Data* (как правило, эта опция установлена по умолчанию).

4. Установите стоимости работ (табл. 7.2). Для назначения стоимости работе следует щелкнуть по ней правой кнопкой мыши и выбрать в контекстном меню *Costs*. Если данные не вводятся, выберите опцию *Override decompositions*.

5. Сгенерируйте отчет *Activity Cost Report* с помощью меню «*Tools/Reports*»: установите флаги у *Activity Name*, *Activity Costs*, *Cost center name*, *Cost center cost*, а для остальных установок – оставьте те, что были заданы по умолчанию. В качестве формата отчета (*Report Format*) следует выбрать «*DDE Table*» (табл. 7.3).

Таблица 7.2

Стоимости работ на диаграмме A2

<i>Activity Name</i>	<i>Cost Center</i>	<i>Cost Center Cost, руб.</i>	<i>Frequency</i>	<i>Duration, день</i>
Отслеживание расписания и управление сборкой и тестированием	Управление	500,00	1,00	1,00
Сборка настольных компьютеров	Рабочая сила	100,00	12,00	1,00
	Компоненты	16000,00		
Сборка ноутбуков	Рабочая сила	140,00	20,00	1,00
	Компоненты	28000,00		
Тестирование компьютеров	Рабочая сила	60,00	32,00	1,00

Таблица 7.3

Отчет Activity Cost Report

<i>Activity Name</i>	<i>Activity Cost, руб.</i>	<i>Cost Center</i>	<i>Cost Center Cost, руб.</i>
Деятельность компании	758 420,00	Компоненты	752 000,00
		Рабочая сила	5 920,00
		Управление	500,00
Продажи и маркетинг	0,00		
Сборка и тестирование компьютеров	758 420,00	Компоненты	752 000,00
		Рабочая сила	5 920,00
		Управление	500,00
Отслеживание расписания и управление сборкой и тестированием	500,00	Управление	500,00
Сборка настольных компьютеров	16 100,00	Компоненты	16 000,00
		Рабочая сила	100,00
Сборка ноутбуков	28 140,00	Компоненты	28 000,00
		Рабочая сила	140,00
Тестирование компьютеров	60,00	Рабочая сила	60,00

6. Сохраните полученный файл *MSWord* в рабочую папку с названием *Activity Cost Report*.

7. Перейдите в меню *Dictionary/UDP Keywords* и в диалоге *UDP Keywords List* внесите ключевые слова *UDP* «Документация», «Информационная система», «Расход ресурсов» (рис. 7.2).

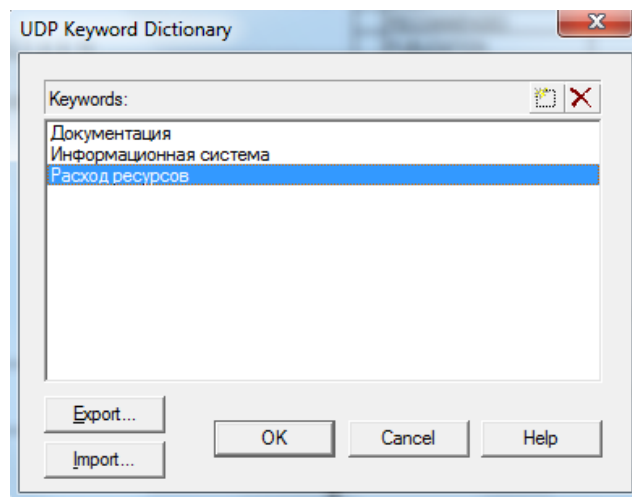


Рис. 7.2. Словарь ключевых слов UDP

8. Откройте *UDP Dictionary (Dictionary/UDP)*. Внесите значения в соответствии с табл. 7.4. Для подключения к *UDP* ключевого слова перейдите к полю *Keyword* и щелкните по полю выбора. Для *UDP* типа *List* необходимо в поле *Value* задать список значений. Для подтверждения ввода данных используйте клавишу табуляции.

Таблица 7.4

Наименование и свойства UDP

Наименование <i>UDP</i>	Тип	Значение	Ключевое слово
Приложения	<i>Text List (Multiple Selection)</i>	Модуль оформления заказов Модуль создания и контроля расписания выполнения работ Модуль учета комплектующих и оборудования Модуль процедур сборки и поиска неисправностей	Информационная система
История изменения	<i>Paragraph Text</i>		Документация
Загрязнение окружающей среды	<i>Text List (Single Selection)</i>	Очень высокое Высокое Среднее Низкое	
Расход электроэнергии	<i>Real Number</i>		Расход ресурсов
Дополнительная документация	<i>Command List</i>	Winword.exe sample2.doc Winword.doc sample3.doc	Документация

После внесенных изменений словарь будет выглядеть так, как показано на рис.7.3.

Name	Definition	UDP Datatype	Value	Keyword
Дополнительная документация		Command List	Edit List	Документация
Загрязнение окружающей среды		Text List (Single selection)	Edit List	
История изменения		Paragraph Text		Документация
Приложения		Text List (Multiple selections)	Edit List	Информационная система
Расход электроэнергии		Real Number		Расход ресурсов
		Text		

Рис. 7.3. Словарь UDP

9. Для назначения *UDP* работе следует щелкнуть по ней правой кнопкой мыши и выбрать в контекстном меню *UDP*. Появляется вкладка *UDP Values* диалога *Activity Properties* (рис. 7.4).

Property	Value
Приложения	Модуль учета комплектующих и оборудования
История изменения	История изменения спецификаций
Загрязнение окружающей среды	Низкое
Расход электроэнергии	10
Дополнительная документация	Winword.exe sample2.doc



Рис. 7.4. Вкладка *UDP Values* диалога *Activity Properties*

10. Внесите значения *UDP* для работ (табл. 7.5).

Таблица 7.5

Значения *UDP* для работ

<i>Activity Name</i>	Дополнительная документация	Приложения	История изменения	Расход электроэнергии	Загрязнение окружающей среды
Отслеживание расписания и управление сборкой и тестированием	Winword.exe sample2.doc	Модуль создания и контроля расписания выполнения работ	История изменения спецификаций	10,00	Низкое
Сборка настольных компьютеров		Модуль учета комплектующих и оборудования. Модуль процедур сборки и поиска неисправностей		20,00	Среднее
Сборка ноутбуков		Модуль учета комплектующих и оборудования. Модуль процедур сборки и поиска неисправностей		25,00	Среднее
Тестирование компьютеров	Winword.exe sample3.doc	Модуль учета комплектующих и оборудования. Модуль процедур сборки и поиска неисправностей		40,00	Среднее

11. После внесения *UDP* типа *Command* или *Command List* щелчок по кнопке  приведет к запуску приложения. Создайте файл «sample2.doc» в рабочем каталоге. Щелкните по кнопке . В результате произойдет запуск приложения. Закройте его.

12. В диалоге *Activity Properties* щелкните по кнопке *Filter*. В появившемся диалоге *Diagram object UDP filter* (рис. 7.5) отключите ключевые слова «Информационная система». В результате в диалоге *Activity Properties* не будут отображаться *UDP* с ключевыми словами «Информационная система».

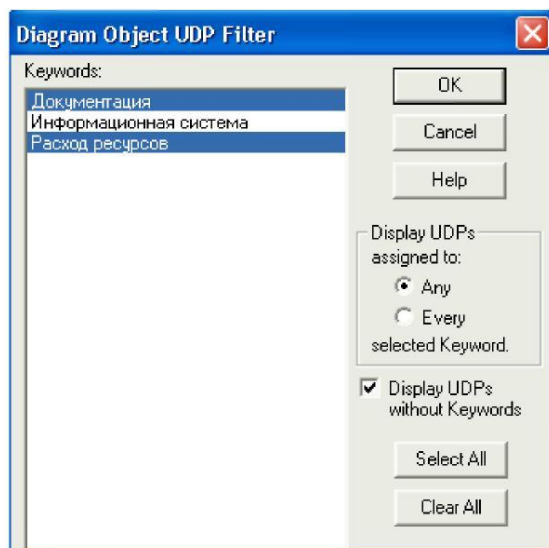


Рис. 7.5. Диалог *Diagram object UDP filter*

13. Создайте отчет по *UDP*. Меню *Tools/Report/Diagram object Report*.
Опции отчета представлены в табл. 7.6.

Таблица 7.6

Опции отчета *Diagram object Report*

Наименование	Значение
<i>Start from Activity</i>	A2
<i>Number of Levels</i>	2
<i>User Defined Properties</i>	Расход электроэнергии
<i>Report Format</i>	<i>DDE Table (MS Word)</i>

14. Щелкните по кнопке *Report*. Сохраните созданный документ *UDP report* в рабочую папку. Содержание отчета отображает табл. 7.7.

Таблица 7.7

UDP Report

<i>Activity Name</i>	Расход электроэнергии
Сборка и тестирование компьютеров	
Отслеживание расписания и управление сборкой и тестированием	10,00
Сборка настольных компьютеров	20,00
Сборка ноутбуков	25,00
Тестирование компьютеров	40,00

Контрольные вопросы

1. Что такое стоимостной анализ?
2. Что означает *UDP*?
3. Каким образом можно сгенерировать отчет?

Практическое занятие № 8

ТЕМА: «РАЗРАБОТКА МОДЕЛИ ПОТОКОВ ДАННЫХ DFD ИНФОРМАЦИОННОЙ СИСТЕМЫ»

Цель занятия: изучить методологии функционального моделирования DFD.

Задача: изучить методологию функционального моделирования потоков данных DFD.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Выполнение декомпозиции модели с помощью нотации DFD.

Теоретические основы занятия

Диаграммы потоков данных (*DFD*) используются для описания документооборота и обработки информации. В соответствии с методологией модель системы определяется как иерархия диаграмм потоков данных, описывающих асинхронный процесс преобразования информации от ее ввода в систему до выдачи пользователю. Диаграммы верхних уровней иерархии (контекстные диаграммы) определяют основные процессы или подсистемы ИС с внешними входами и выходами. Они детализируются при помощи диаграмм нижнего уровня. Такая декомпозиция продолжается, создавая многоуровневую иерархию диаграмм, до тех пор, пока не будет достигнут такой уровень декомпозиции, на котором процесс становится элементарным и детализировать их далее невозможно. Их можно использовать как дополнение к модели *IDEF0* для более наглядного отображения текущих операций документооборота в корпоративных системах обработки информации. *DFD* описывает:

- системы/подсистемы;
- функции обработки информации (работы);
- документы (стрелки, *arrow*), объекты, сотрудников или отделы, которые участвуют в обработке информации;
- внешние ссылки (*external references*), которые обеспечивают интерфейс с внешними объектами, находящимися за границами моделируемой системы;
- таблицы для хранения документов (хранилище данных, *data store*).

В *Brwin* для построения диаграмм потоков данных используется нотация Гейна–Сарсона.

Для того чтобы дополнить модель *IDEF0* диаграммой *DFD*, нужно в процессе декомпозиции в диалоге *Activity Box Count* кликнуть по радиокнопке *DFD*.

В палитре инструментов на новой диаграмме *DFD* появляются новые кнопки:

- добавить в диаграмму внешнюю ссылку (*External Reference*). Внешняя ссылка является источником или приемником данных извне модели;
- добавить в диаграмму хранилище данных (*Data store*). Хранилище данных позволяет описать данные, которые необходимо сохранить в памяти прежде, чем использовать в работах;

- ссылка на другую страницу. В отличие от *IDEF0* инструмент *offpage reference* позволяет направить стрелку на любую диаграмму (а не только на верхний уровень).

В отличие от стрелок *IDEF0*, которые представляют собой жесткие взаимосвязи, стрелки *DFD* показывают, как потоки данных перемещаются и изменяются от одной работы к другой. Это представление потоков совместно с хранилищами данных и внешними сущностями делает модели *DFD* более похожими на физические характеристики системы – движение объектов (*data flow*), хранение объектов (*data stores*), поставку и распространение объектов (*external entities*).

DFD рассматривает систему как совокупность предметов. Контекстная диаграмма часто включает работы и внешние ссылки. Работы обычно именуются по названию системы, например «Система обработки информации». Включение внешних ссылок в контекстную диаграмму не отменяет требования методологии четко определить цель, область и единую точку зрения на моделируемую систему.

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

Работы. В *DFD* работы представляют собой функции системы, преобразующие входы в выходы. Хотя работы изображаются прямоугольниками со скругленными углами, смысл их совпадает со смыслом работ *IDEF0* и *IDEF3*. Так же как работы *IDEF3*, они имеют входы и выходы, но не поддерживают управления и механизмы, как *IDEF0*.

Внешние сущности. Внешние сущности изображают входы в систему и/или выходы из системы. Внешние сущности изображаются в виде прямоугольника с тенью и обычно располагаются по краям диаграммы. Одна внешняя сущность может быть использована многократно на одной или нескольких диаграммах. Обычно такой прием используют, чтобы не рисовать слишком длинных и запутанных стрелок.

Стрелки (потоки данных). Стрелки описывают движение объектов из одной части системы в другую. Поскольку в *DFD* каждая сторона работы не имеет четкого назначения, стрелки могут подходить к любой грани и выходить из любой грани прямоугольника работы. В *DFD* также применяются двунаправленные стрелки для описания диалогов типа «команда-ответ» между работами, между работой и внешней сущностью и между внешними сущностями.

Хранилище данных. В отличие от стрелок, описывающих объекты в движении, хранилища данных изображают объекты в покое. В материальных системах хранилища данных изображаются там, где объекты ожидают обработки, например в очереди. В системах обработки информации хранилища данных являются механизмом, который позволяет сохранить данные для последующих процессов.

Слияние и разветвление стрелок. В *DFD* стрелки могут сливаться и разветвляться, что позволяет описать декомпозицию стрелок. Каждый новый сегмент сливающейся или разветвляющейся стрелки может иметь собственное имя.

Построение диаграмм *DFD*. Диаграммы *DFD* могут быть построены с использованием традиционного структурного анализа подобно тому, как строятся диаграммы *IDEF0*. Сначала строится физическая модель, отображающая текущее состояние дел. Затем эта модель преобразуется в логическую модель, которая отображает требования к существующей системе. После этого строится модель, отображающая требования к будущей системе. И наконец, строится физическая модель, на основе которой должна быть построена новая система.

Альтернативным подходом является подход, популярный при создании программного обеспечения, называемый событийным разделением (*event partitioning*), в котором различные диаграммы *DFD* выстраивают модель системы. Логическая модель строится как совокупность работ и документирования того, что они (эти работы) должны делать. Затем модель окружения (*environment model*) описывает систему как объект, взаимодействующий с событиями из внешних сущностей. Модель окружения обычно содержит описание цели системы, одну контекстную диаграмму и список событий. Контекстная диаграмма содержит один прямоугольник работы, изображающий систему в целом, и внешние сущности, с которыми система взаимодействует. Наконец, модель поведения (*behavior model*) показывает, как система обрабатывает события. Эта модель состоит из одной диаграммы, в которой каждый прямоугольник изображает каждое событие из модели окружения. Хранилища могут быть добавлены для моделирования данных, которые необходимо запоминать между событиями. Потоки добавляются для связи с другими элементами, и диаграмма проверяется с точки зрения соответствия модели окружения.

Полученные диаграммы могут быть преобразованы с целью более наглядного представления системы, в частности работы на диаграммах могут быть декомпозированы.

Нумерация объектов. В *DFD* номер каждой работы может включать префикс, номер родительской работы (*A*) и номер объекта. Номер объекта – это уникальный номер работы на диаграмме. Например, работа может иметь номер *A.12.4*. Уникальный номер имеют хранилища данных и внешние сущности независимо от их расположения на диаграмме. Каждое хранилище может иметь префикс *D* и уникальный номер, например *D5*. Каждая внешняя сущность – префикс *E* и уникальный номер, например *E5*.

Экспериментальная часть занятия

При оформлении заказа важно проверить, существует ли такой клиент в базе данных и, если не существует, внести его в базу данных и затем оформить заказ. Оформление заказа начинается со звонка клиента. В процессе оформления заказа база данных клиентов может просматриваться и редактироваться.

Заказ должен включать как информацию о клиенте, так и информацию о заказанных продуктах. Оформление заказа подразумевает чтение и запись информации о прочих заказах.

В процессе декомпозиции согласно правилам *DFD* необходимо преобразовать граничные стрелки во внутренние, начинающиеся и заканчивающиеся на внешних ссылках.

1. Декомпозируйте работу «Оформление заказов» на диаграмме *A2*. В диалоге *Activity Box Count* выберите количество работ 2 и нотацию *DFD* (рис. 8.1).

2. Щелкните по *OK* и внесите в новую диаграмму *DFD A22* имена работ «Проверка и внесение клиента» и «Внесение заказа».

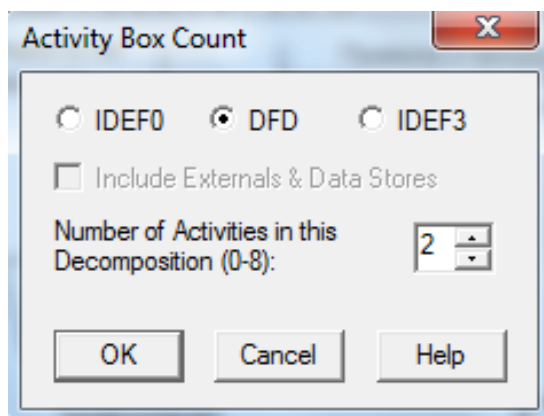



Рис. 8.1. Выбор нотации *DFD* в диалоге *Activity Box Count*

3. Используя кнопку  на палитре инструментов, внесите хранилища данных «Список клиентов», «Список продуктов» и «Список заказов».

4. Удалите граничные стрелки с диаграммы *DFD A22*.

5. Используя кнопку  на палитре инструментов, внесите внешнюю ссылку «Звонки клиентов».

6. Создайте внутренние ссылки согласно рис. 8.2. При переименовании стрелок используйте словарь.

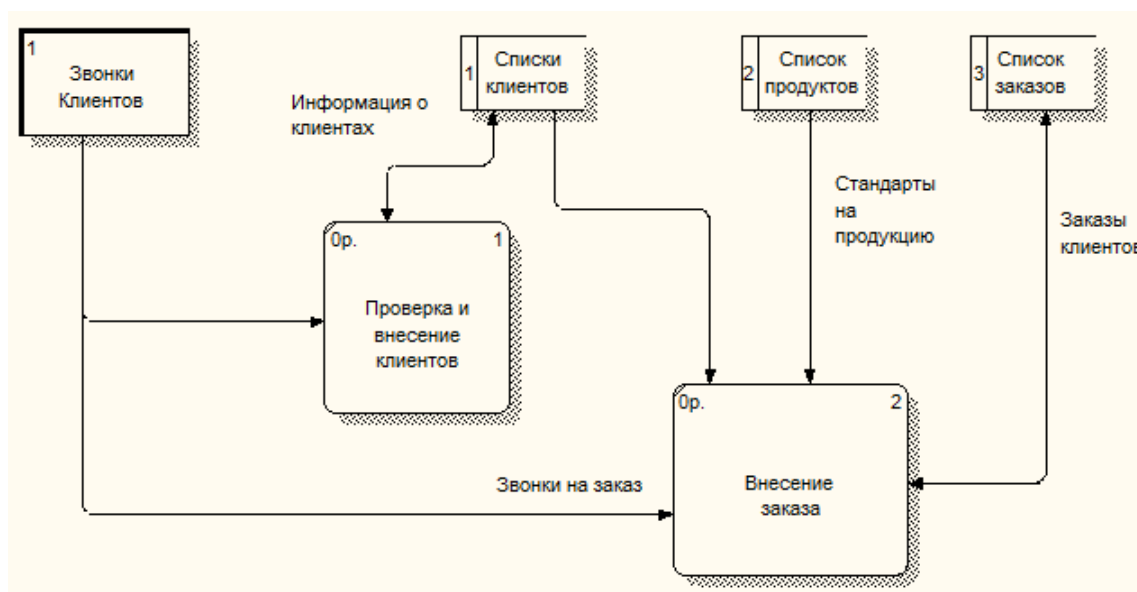


Рис. 8.2. Диаграмма *A22* «Оформление заказов»

Обратите внимание, что стрелки «Информация о клиентах» и «Заказы клиентов» двунаправленные. Для этого щелкните правой кнопкой по стрелке, выберите в контекстном меню пункт *Style* выберите опцию *Bidirectional*.

7. На родительской диаграмме *A2* туннелируйте (*Change to Tunnel*) стрелки, подходящие и исходящие из работы «Оформление заказов» (рис. 8.3).

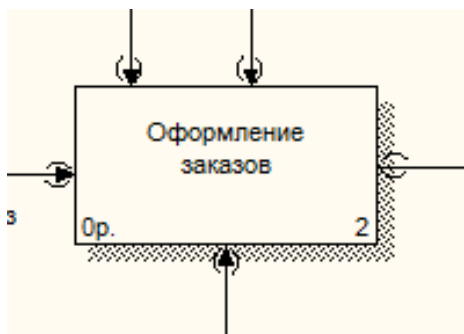




Рис. 8.3. Работа «Оформление заказов» на диаграмме *A2*

Некоторые стрелки с диаграмм *IDEF0* и *DFD* могут показываться на диаграмме *DFD*. Для отображения таких стрелок используется инструмент *Off-Page Reference*.

8. Декомпозируйте работу «Исследование рынка» на диаграмме *A2* на диаграмму *DFD*. Удалите граничные стрелки. Создайте работы «Разработка прогнозов продаж», «Разработка маркетинговых материалов» и «Привлечение новых клиентов».

9. Используя кнопку  на палитре инструментов, внесите ранее созданные хранилища данных, используя выпадающий список «Список клиентов», «Список продуктов» и «Список заказов».

10. Добавьте две внешние ссылки «Маркетинговые материалы» и «Прогноз продаж», используя кнопку .

11. Свяжите объекты диаграммы *DFD* стрелками, как показано на рис. 8.4.

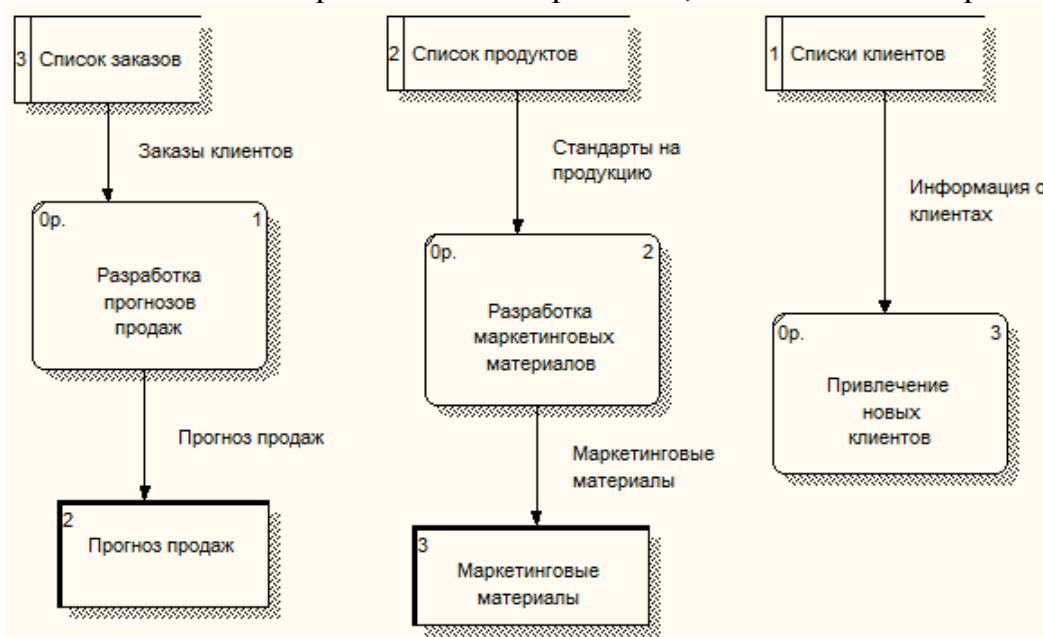


Рис. 8.4. Диаграмма *A23* «Исследование рынка»

12. На родительской диаграмме *A2* туннелируйте (*Change to Tunnel*) стрелки, подходящие и исходящие из работы «Исследование рынка».

13. В случае внесения новых клиентов в работе «Проверка и внесение клиента» на диаграмме *A22* «Оформление заказов» информация должна направляться к работе «Привлечение новых клиентов» диаграммы *A23* «Исследование рынка». Для этого необходимо использовать инструмент *Off-Page Reference*. На диаграмме *A22* «Оформление заказов» создайте новую граничную стрелку, исходящую от работы «Проверка и внесение клиента», и назовите ее «Информация о новом клиенте» (рис. 8.5).

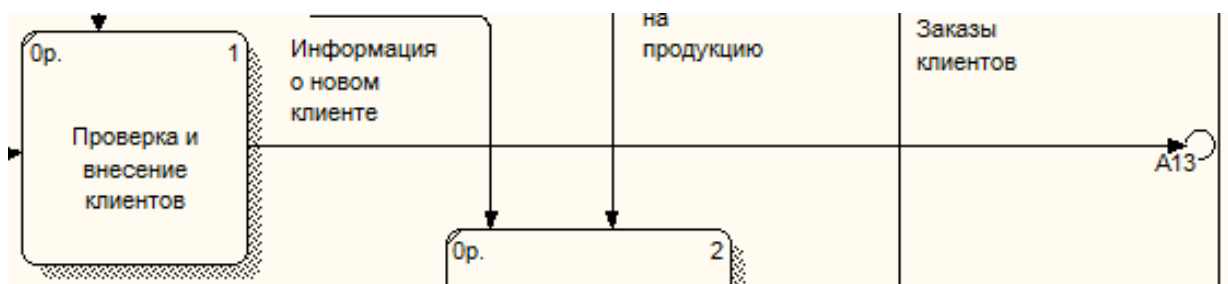


Рис. 8.5. Стрелка «Информация о новом клиенте» на диаграмме *A12*

14. Перейдите в меню *Model/Model Properties*, далее – во вкладку *Display*. Установите опцию *Off-Page Reference label - Node number*.

15. Правой кнопкой щелкните по наконечнику стрелки и выберите в меню *Off-Page Reference*. В появившемся диалоге *Off-Page Arrow Reference* (рис. 8.6) выберите в качестве диаграммы *A13* «Исследование рынка». В качестве *Destination border* выберите *Input*. Нажмите кнопку «*Ok and Go To Diagram*».

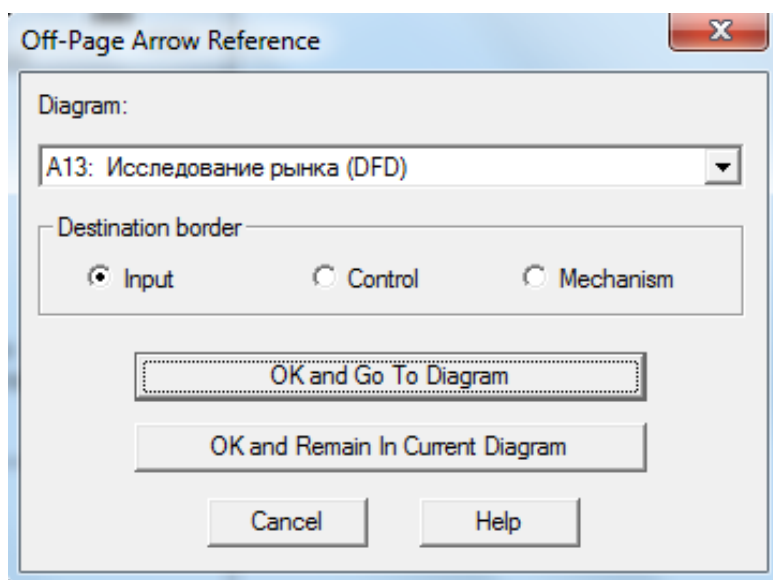


Рис. 8.6. Диалог *Off-page arrow reference*

16. На диаграмме *A13* «Исследование рынка» направьте стрелку «Информация о новом клиенте» на вход работы «Привлечение новых клиентов». Результат представлен на рис. 8.7.

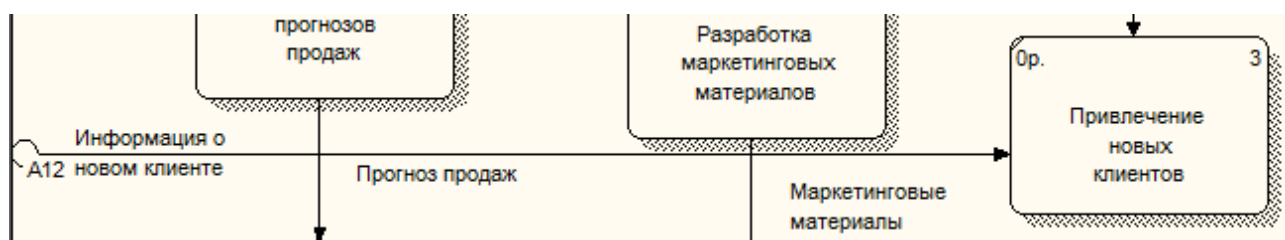


Рис. 8.7. Межстраничная ссылка на диаграмме A13

Контрольные вопросы

1. Чем отличаются диаграммы *DFD* от диаграмм *IDEF0* и *IDEF3*?
2. Что такое хранилище данных?
3. Какие процессы описывают диаграммы *DFD*?
4. Как изображаются работы на диаграммах *DFD*?
5. Что такое межстраничная ссылка?

Практическое занятие № 9

ТЕМА: «РАЗРАБОТКА СТРУКТУРЫ ДАННЫХ МЕТОДОМ ER-ДИАГРАММ»

Цель занятия: изучить методологию IDEF1X построения структурных диаграмм данных «сущность-связь».

Задача: изучить методологию IDEF1X построения структурных диаграмм данных «сущность-связь».

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Построение модели логического уровня диаграммы «сущность-связь».
3. Уточнение атрибутов сущностей, определение ключевых атрибутов.
4. Определение зависимости между сущностями, их типа, мощности связи.

Теоретические основы занятия

Большинство автоматизированных информационных систем должны обеспечивать обработку данных. Для этого в их состав вводится функциональный блок ведения базы данных, рассмотренный в предыдущей работе. Проектирование структуры базы данных должно быть произведено с учетом требований по обеспечению логической и физической целостности данных, защите информации, возможности восстановления и расширения, совместимости с другими системами. Модель данных может быть иерархической, сетевой, хронологической или реляционной. Учитывая распространенность современных реляционных систем управления баз данных (СУБД) Oracle Database, Microsoft SQL Server, InterBase, Sybase, IBM DB2 и т.д., в данной работе рассматривается реляционная модель логического представления данных.

Отметим, что созданная база данных считается доступной из любого функционального блока автоматизированной информационной системы.

Проектирование базы данных может быть разделено на этапы концептуального, логического и физического проектирования.

ERwin – средство моделирования баз данных, в котором используется методология информационного моделирования на основе ER-диаграмм (диаграмм сущность-связь) IDEF1X. ERwin реализует проектирование схемы баз данных и генерацию ее описания на языке целевой СУБД. ERwin имеет два уровня представления данных – логический и физический.

ERwin имеет несколько уровней отображения диаграммы: сущностей, атрибутов, определений, первичных ключей и иконок. Переключиться между первыми тремя уровнями можно с использованием кнопок панели инструментов. Переключиться на другие уровни отображения можно при помощи контекстного меню, в котором следует выбрать пункт *Display Level* и затем необходимый уровень отображения.

В терминологии ERwin для логической модели используются три уровня представления, отличающихся степенью детализации данных:

- *диаграмма сущность-связь* – модель данных верхнего уровня. Она включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области, может включать связи многие-ко-многим и не включать описания ключей;

- *модель данных, основанная на ключах* – более подробное представление данных. Она включает описание всех сущностей и первичных ключей и предназначена для представления структуры данных и ключей, которые соответствуют предметной области;




- *полная атрибутивная модель* – наиболее детальное представление структуры данных: представляет данные в третьей нормальной форме и включает все сущности, атрибуты и связи.


Основные компоненты диаграммы Erwin следующие:

- *сущность*. Она определяется как объект, событие или концепция, информация о котором должна сохраняться. Сущности должны иметь наименование с четким смысловым значением, именоваться существительными в единственном числе, не носить технических наименований и быть достаточно важными, чтобы их моделировать. Каждый экземпляр сущности должен быть уникальным;

- *атрибут*. Он хранит информацию об определенном свойстве сущности. Атрибут или группа атрибутов, которые однозначно идентифицируют сущность, называются первичным ключом (*Primary Key*). Для описания атрибутов следует в контекстном меню выбрать пункт *Attribute Editor*. Атрибуты должны именоваться в единственном числе и иметь четкое смысловое значение;

- *связь*. Она является логическим соотношением между сущностями. Каждая связь должна именоваться глаголом или глагольной фразой (*Relationship Verb Phrases*). По умолчанию имя связи на диаграмме не показывается. Для отображения имени связи следует в пункте *Display Options/Relationship* контекстного меню диаграммы включить опцию *Verb Phrase*.

На логическом уровне можно установить идентифицирующую связь один-ко-многим , связь многие-ко-многим  и неидентифицирующую связь один-ко-многим . Связь сущности с другими сущностями автоматически определяет ее тип (зависимая и независимая).

Идентифицирующая связь устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями. Зависимая сущность не может существовать самостоятельно и изображается прямоугольником со скругленными углами . При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи

называется миграцией атрибутов. В дочерней сущности новые атрибуты помечаются как внешний ключ (*FK*).

Неидентифицирующая связь служит для связывания независимых сущностей. При этом атрибуты первичного ключа родительской сущности мигрируют в состав неключевых компонентов дочерней сущности.

Редактирование свойств связи осуществляется с помощью пункта *Relationship Editor* контекстного меню диаграммы. Мощностъ связи (*Cardinality*) служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней. На вкладке *Definition* можно дать полное определение связи. На вкладке *Rolename/RI Actions* можно задать имя роли и правила ссылочной целостности. Имя роли (функциональное имя) – это синоним атрибута внешнего ключа, который показывает, какую роль играет атрибут в дочерней сущности. Имя роли используется, когда несколько атрибутов одной сущности имеют одинаковую область значений, но разный смысл или в случае рекурсивных связей (*fish hook*). Рекурсия может быть иерархической или сетевой.

Правила ссылочной целостности (*Referential Integrity*) – логические конструкции (условия, предикаты), которые выражают бизнес-правила использования данных и представляют собой правила правильного выполнения операций добавления, замены и удаления данных. При генерации схемы БД на основе логической модели будут сгенерированы правила ссылочной целостности, которые должны быть предписаны для каждой связи, и триггеры (программные реализации правил), обеспечивающие ссылочную целостность.

Иерархия наследования (категорий) – особый тип объединения сущностей, обладающих общими характеристиками (категориальных сущностей) – обобщение. Для каждой категории указывается дискриминатор – атрибут родителя, который показывает, как отличить одну категориальную сущность от другой. Для редактирования категорий нужно выбрать символ категории и выбрать в контекстном меню пункт *Subtype Relationship Editor*. В диалоге *Subtype Relationship* можно указать атрибут-дискриминатор категории (*Discriminator Attribute Choice*) и тип категории – неполная/полная (*Complete/Incomplete*). Неполная категория имеет неполное перечисление обобщаемых частных примеров.

Различают два уровня физической модели: трансформационную модель (*Transformation Model*) и модель СУБД (*DBSM Model*). Физическая модель содержит всю информацию, необходимую для реализации конкретной базы данных. Трансформационная модель содержит информацию для реализации отдельного проекта, который может быть частью общей информационной системы и описывать подмножество предметной области. ERwin поддерживает ведение отдельных проектов, позволяя проектировщику выделять подмножество модели в виде предметных областей (*Subject Area*). Модель СУБД автоматически генерируется из трансформационной модели и является точным отображением системного каталога СУБД.

Физический уровень представления модели зависит от выбранного сервера, предполагается клиент-серверная архитектура АС.

Для выбора СУБД служит редактор *Target Server*, меню *Server/Target Server...* – доступно только на физическом уровне. ERwin поддерживает более 20 реляционных и нереляционных баз данных.

Представления – это объекты базы данных, данные в которых не хранятся постоянно, как в таблице, а формируются динамически при обращении к представлению. Палитра инструментов ERwin на физическом уровне содержит кнопки внесения представлений и установления связей между таблицами и представлениями.

При генерации схемы физической базы данных ERwin автоматически создает отдельный индексный файл на основе первичного ключа каждой таблицы, а также на основе всех альтернативных ключей, внешних ключей и инверсионных входов, поскольку эти столбцы наиболее часто используются для поиска данных. Изменить характеристики существующего индекса или создать новый можно с помощью контекстного меню таблицы *Index* в редакторе *Index Editor*.

Для создания триггера служит редактор *Table Trigger Editor* (вызывается с помощью кнопки *Table Trigger* диалога *Table Trigger Viewer*). Триггеры и хранимые процедуры – это именованные блоки кода SQL, которые заранее откомпилированы и хранятся на сервере для того, чтобы быстро производить выполнение запросов, проверку достоверности данных и выполнять другие часто вызываемые функции. Хранимые процедуры могут вызываться из клиентского приложения или другой хранимой процедуры.


Триггер – это особый вид хранимой процедуры, выполняемый автоматически при добавлении, изменении или удалении строк в существующие таблицы. Для генерации триггеров ERwin использует механизм шаблонов – специальных скриптов, использующих макрокоманды. Шаблоны триггеров ссылочной целостности, генерируемые ERwin, по умолчанию можно изменить, кроме того, можно переопределить как триггеры для конкретной связи, так и шаблоны во всей модели в целом. Для создания или редактирования хранимой процедуры следует выбрать в контекстном меню пункт *Table Editor/Stored Procedure*. ERwin позволяет связывать хранимые процедуры не только с отдельными таблицами, но и со всей моделью.

Экспериментальная часть занятия

1. Запустите приложение ERwin. Выберите в диалоговом окне запроса *Create a new mode*.

2. Создайте модель. В меню ERwin выберите *File/New*. Появится диалог *Create Model Select Template*. В качестве типа для новой модели выберите *Logical/Physical* и тип базы данных *SQL Server 2005/2008* для физической модели.

3. Сохраните модель с помощью меню *File/Save As dialog*. Введите имя для модели, например «My_Model».

4. Добавьте сущность. Активируйте режим добавления сущности с помощью кнопки панели инструментов . Затем нажмите левой клавишей мыши в

любом месте диаграммы для установки первой сущности. По умолчанию сущность будет названа «E/1». Добавьте еще две сущности (смотри рис. 9.1).

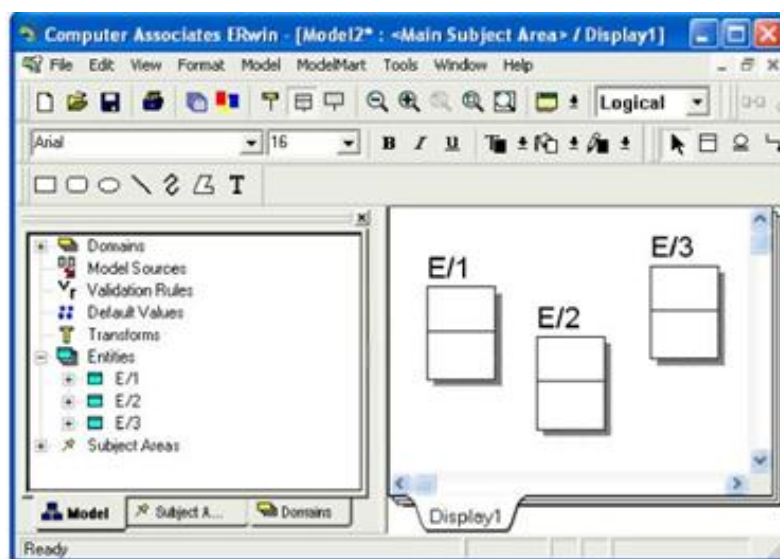


Рис. 9.1. Модель с добавленными объектами

5. Сохраните модель. Сущности, которые были созданы в окне диаграммы, появятся в *Model Explorer*.

6. Переименуйте сущность. Можно назвать сущность непосредственно в окне диаграммы или в *Model Explorer*.

Первый способ. В окне диаграммы расположите сущность с именем «E/1». Сделайте двойной щелчок мыши на имени сущности. Появится блок редактирования вокруг имени. Наберите фразу «Customer» над именем сущности и нажмите левую клавишу мыши вне блока редактирования.

Второй способ. В *Model Explorer* щелкните правой кнопкой мыши по фразе «E/2» и из контекстного меню выберите пункт *Rename* (переименовать). Введите фразу «Order». Щелкните левой клавишей мыши вне блока редактирования.

Одним из вышеприведенных способом переименуйте последнюю сущность «E/3» на «Product». Оцените результат (рис. 9.2) и сохраните модель.

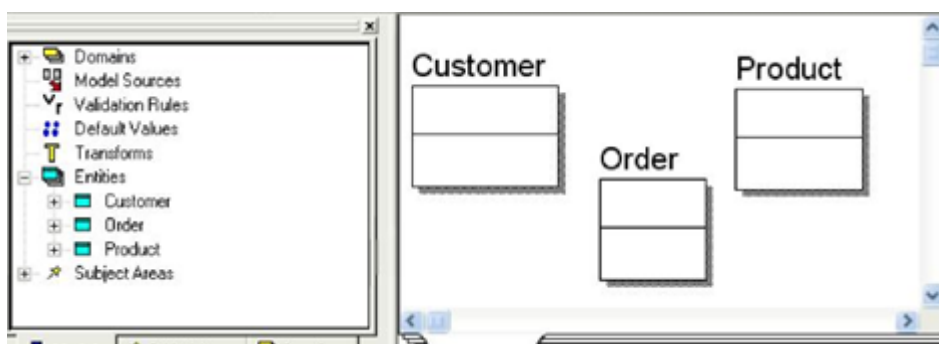


Рис. 9.2. Логическая модель из трех сущностей

7. Добавьте атрибуты. Можно добавлять атрибуты к сущностям непосредственно в окне диаграммы или в *Model Explorer*. Рассмотрим два способа.

Первый способ. В окне диаграммы щелкните по «Customer» Появится редактор атрибутов сущности. Добавьте первый атрибут, являющийся первичным ключом (рис. 9.3) и закройте окно кнопкой *OK*.

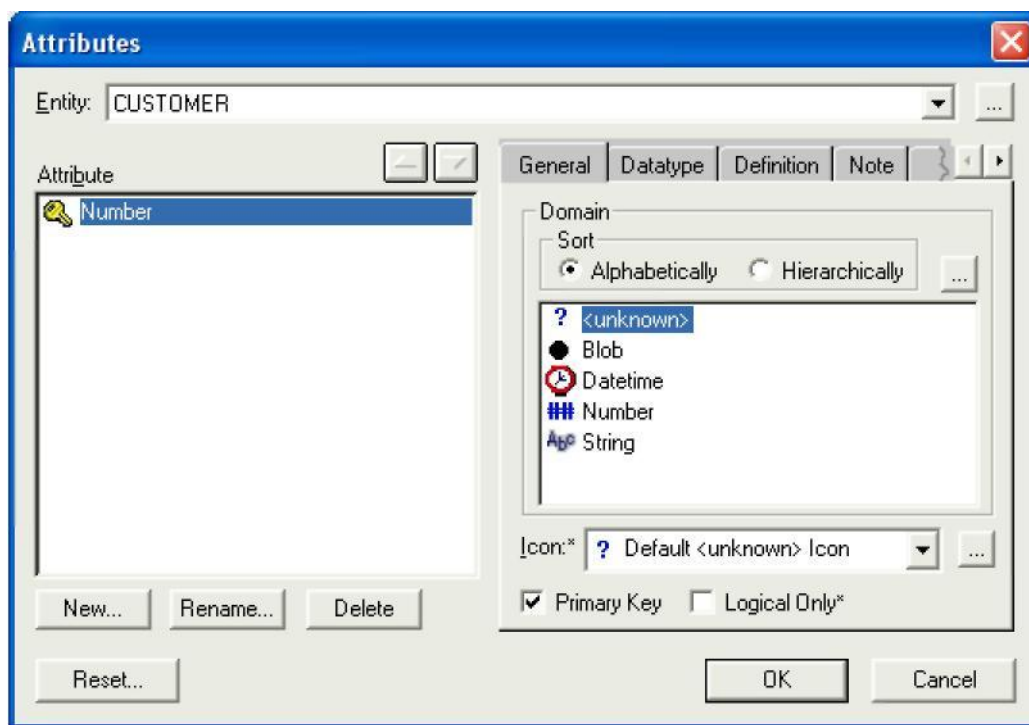


Рис. 9.3. Добавление нового атрибута первым методом

Второй способ. Добавьте атрибут в *Model Explorer*. Для этого щелкните в плюсе около «Customer». Это расширит объектный список. Щелкните правой кнопкой мыши по *Attributes* и выберите *New* из контекстного меню. Атрибут «New Attribute» переименуйте на «Name» (рис. 9.4).

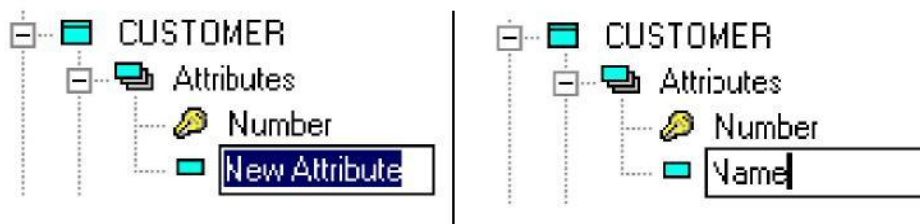


Рис. 9.4. Добавление нового атрибута вторым методом

8. Добавьте к сущности «Customer» атрибуты «Price», «Phone» «Number». Определите атрибут «Price» как первичный ключ, задайте новый тип данных, для этого откройте вкладку *Datatype* и измените тип данных на любой из выпадающего списка.

9. Удалите атрибут «Price» одним из приведенных ниже способов.

Первый способ. В *Model Explorer* выделите атрибут «Price» и нажмите кнопку *Del*, подтвердите удаление атрибута.

Второй способ. В окне диаграммы выделите атрибут «Price» (при первом щелчке выделится сущность, а при втором – нужный атрибут) и нажмите кнопку *Del*, подтвердите удаление атрибута.

10. Выполните описание для сущностей. Зайдите в редактор *Entities*. Выберите сущность из списка *Entity*, откройте закладку *Definition* и введите описания для сущностей: «Customer» – «Таблица клиентов», «Order» – «Заказы» и «Product» – «Продукты».

11. В панели инструментов кликните по одной из кнопок определяющих средство отношения (*Identifying Relationship*). В окне диаграммы добавьте определяющее отношение между «Customer» и «Order», щелкнув по объекту «Customer», чтобы назначить объект-родитель. Затем щелкните по «Order», чтобы определить его как объект-потомок.

12. На палитре инструментов кликните по кнопке «не идентифицирующего» отношения (*Non-identifying*). Для создания отношения между «Order» и «Product» сначала щелкните один раз левой клавишей мыши по сущности «Order», а потом по сущности «Product». Модель должна выглядеть подобно нижеприведенной модели: на рис. 9.5.

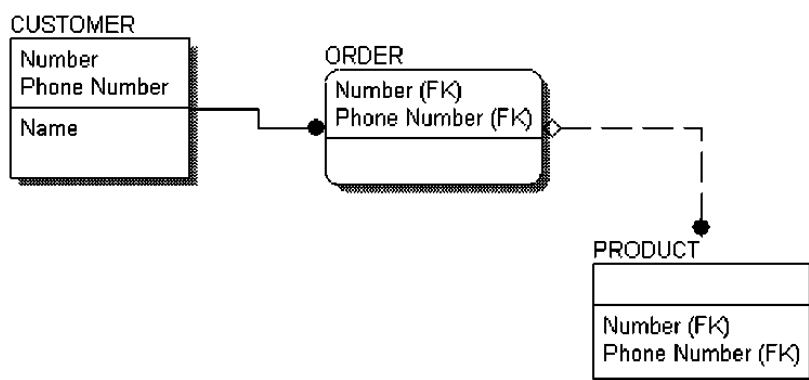


Рис. 9.5. Модель после внесения связей

13. Свяжите внешние ключи с соответствующими именами ролей: «Номер ключа клиента» и «Номер телефона клиента» для определяющей связи (закладка *Rolename* редактора *Relationships*). Включите отображение базового имени атрибута (*Entity Display/Rolename/Attribute*), как показано на рис. 9.6.

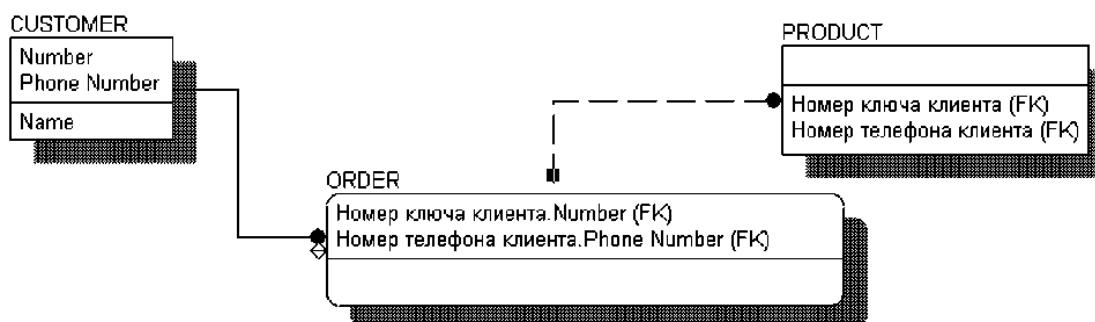


Рис. 9.6. Определение имени роли для внешних ключей

14. Задайте глагольную фразу для связей: «<Клиент>(Customer) размещает <Заказ>(Order)» и «<Заказ> выписывает <Товар>(Product)».

15. Установите для двух связей «Customer»—«Order» и «Order»—«Product» следующий вид мощности: каждая родительская сущность связана с 0, 1 и более экземпляров дочерней сущности.

16. Поменяйте местами атрибуты «Name» и «Phone Number» сущности «Customer». Сохраните модель.

Контрольные вопросы

1. Основная цель использования *ERWin*.
2. Какие бывают виды представления модели данных?
3. Что такое сущность?
4. Что такое атрибут?
5. Что такое связь?
6. Какие существуют виды связей?
7. Что такое мощность связи?
8. Что такое роль атрибута?
9. Что такое внешний ключ?

Практическое занятие № 10

ТЕМА: «ПРЯМОЕ И ОБРАТНОЕ ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ»

Цель занятия: получить навыки в генерации схемы базы данных, файла сценария.

Задача: получить навыки в генерации схемы базы данных и файла сценария.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка модели физического уровня, выбор типа СУБД.
3. Генерация SQL-сценария для создания структуры базы данных.
4. Установление соответствия между потоками данных диаграммы DFD и сущностями диаграммы IDEF1X.

Теоретические основы занятия

Прямое и обратное проектирование базы данных

ERwin поддерживает обратное проектирование существующих баз данных и обеспечивает физическую и логико-физическую модель данных, поэтому можно поддерживать созданную базу данных или осуществлять передачу от текущего целевого сервера к другому.

Процесс генерации физической схемы базы данных из логической модели данных называется прямым проектированием (*Forward Engineering*).

Процесс генерации логической модели из физической базы данных называется обратным проектированием (*Reverse Engineering*). После создания модели можно произвести обратное проектирование структуры базы данных, а затем легко перенести его в другой формат базы данных.

Когда есть физическая модель данных, ERwin автоматически генерирует схему для целевого сервера при формировании модели. Обратное проектирование является процессом, который используется ERwin, для передачи схемы от модели данных к целевому серверу. Когда пересылается проект модели данных, можно сгенерировать файл сценария (*script*), который будет использоваться для корректировки базы данных. Корректировка базы данных будет вестись с помощью административного средства (*database administration tool*) или пересылкой проекта непосредственно для подсоединения к базе данных каталога.

Перед пересылкой проекта можно рассмотреть схему, являющуюся базовым текстовым представлением базы данных объектов, которые будут созданы в базе данных из сценария (*script*). ERwin использует язык определения данных (*DDL*) для целевой базы данных, чтобы записывать *script*. При каждом добавлении объекта или свойства в модель данных ERwin автоматически корректирует файл сценария для отображения изменений в модели данных.

ERwin может произвести обратное проектирование существующей базы данных, автоматически считывая определение схемы, и создать диаграмму

модели данных. При обратном проектировании базы данных ERwin автоматически генерирует расположение объектов на диаграмме по умолчанию. После того как диаграмма будет сгенерирована в результате обратного проектирования, можно, используя инструменты и редакторы ERwin, добавлять новые объекты, создавать системную документацию и перепроектировать структуру базы данных, основываясь на изменениях технических и (или) организационных требований.

Разные СУБД на разных уровнях обеспечивают синтаксическую поддержку связей, индексов, ссылочной целостности и других свойств.

ERwin автоматически создает новое окно диаграммы главной области и показывает на экране схему в виде графической модели данных.

Помимо импорта информации, явным образом определенной в физической схеме, ERwin извлекает значительный объем информации из схемы и встраивает ее в диаграмму в процессе создания новой модели данных путем обратного проектирования.

Запуск процесса проектирования данных. При создании модели данных обратным проектированием можно ускорять проектировку новой модели данных с последующей поставкой новых систем.

ERwin позволяет переделать реальную схему базы данных, которая преобразована ERwin в графическое представление структур базы данных. ERwin создает модели данных непосредственно из базы данных.

После подсоединения к базе данных ERwin создает активное соединение в двух направлениях с системным каталогом конкретной базы данных. Это соединение позволяет производить прямое и обратное проектирование схемы непосредственно в каталог базы данных (не требуется запускать скрипт языка определения данных, как отдельный процесс). Можно синхронизировать изменения, вносимые в модель ERwin, непосредственно с системным каталогом. При синхронизации ERwin запрашивает системный каталог и сообщает о различиях, найденных между пользовательской базой данных и ERwin.

Можно сгенерировать полную схему базы данных, используя имена таблиц, имена колонок, имена физических связей и типов данных, присвоенных в ERwin. ERwin может также генерировать триггеры ссылочной целостности, хранимые процедуры, индексы, домены и другие ограничения, если они поддерживаются пользовательской СУБД.

Генерация схемы базы данных. ERwin предоставляет две возможности при генерации схемы базы данных:

- подсоединение ERwin непосредственно к системному каталогу базы данных и генерация схемы за один шаг;
- генерацию скрипта *ASCII DDL* (на языке определения данных). Скрипт *DDL* должен быть выполнен на сервере для генерации схемы, и это должно быть отдельное действие.

Редактор *<DB> Schema Generation* позволяет выбрать те определения физических объектов ERwin, то есть таблицы, индексы, триггеры, хранимые процедуры и т.д., которые нужно включить в генерируемую схему (см. заклад-

ку *Options*). Редактор открывается из меню *Tools* командой *Forward Engineer / Schema Generation*. Возможности, доступные в редакторе, различаются в зависимости от того, какие возможности поддерживаются пользовательской СУБД.

Для использования части сущностей текущей области с целью генерации схемы следует воспользоваться кнопкой *Filter* в редакторе.

Кнопки *Preview*, *Print* и *Report* в нижней части редактора позволяют просматривать отчет на экране, распечатывать его или сохранять на диске в текстовом файле.

Кнопка *Generate* в редакторе *Schema Generation Report* служит для запуска процесса генерации схемы. При нажатии на кнопку *Generate* ERwin выводит на экран диалог *<DB> Connection*, который позволяет подсоединиться к базе данных и связать ERwin с системным каталогом базы данных.

Для генерации схемы из окна *Preview* выполните следующие действия:

1. Нажмите кнопку *Preview*, расположенную в нижней части редактора, для входа в окно *Schema Generation Preview*.

2. По умолчанию ERwin генерирует всю схему полностью. Для того чтобы сгенерировать часть схемы, нажмите левую кнопку мыши и, не отпуская ее, передвигайте мышь вниз, выделяя текст схемы, который следует выбрать. Отпустите кнопку мыши, когда будет достигнут конец генерируемой части.

3. Нажмите кнопку *Generate*, расположенную в нижней части окна *Preview*. ERwin генерирует схему. Если при генерации схемы возникнет ошибка, ERwin выдает сообщение об ошибке. Чтобы игнорировать ошибку и продолжить работу по генерации схемы, нажмите кнопку *Continue*.

4. Чтобы остановить процесс генерации схемы, нажмите кнопку *Abort*. ERwin возвращается в редактор *<DB> Schema Generation*.

5. После того как ERwin завершит процесс генерации схемы, он возвращается в окно *Preview*. Для выхода из окна *Preview* в редактор *<DB> Schema Generation* нажмите кнопку *Close*.

6. Чтобы распечатать отчет по схеме в том виде, в котором он демонстрируется в окне «*Preview*», нажмите кнопку *Print*. ERwin закрывает редактор *<DB> Schema Generation* и распечатывает отчет. При редактировании отчета в окне *Preview* ERwin распечатывает отчет с изменениями.

7. Для сохранения файла отчета по схеме можно нажать кнопку *Report*. ERwin открывает окно-диалог *Generate <DB> Schema Report*.

Примечание. Если поставить метку в окне *Stop If Failure*, ERwin приостановит работу в случае ошибки. Нажатием кнопки *Continue* продолжается генерация схемы. Нажатием кнопки *Abort* отменяется генерация схемы. Можно отредактировать коды в окне *Preview*, а затем нажать кнопку *Generate*, чтобы сгенерировать отредактированную версию.

Сохранить файл отчета по схеме можно с помощью нажатия на кнопку *Report*. В диалоге *Generate <DB> Schema Report* выбрать тип файла отчета.

При работе в редакторе *Schema Generation* доступны различные режимы, в зависимости от СУБД. Поддерживаемые режимы находятся в соответствующем

групповом окне. Список групповых окон следующий: *Referential Integrity, Trigger, Table, Index, Column, Schema u Other Options*.

Referential Integrity – это режимы ссылочной целостности (RI), они позволяют указывать, как поступать со связанными записями, если значение в поле ключа изменяется или удаляется. Возможные режимы следующие:

- *Primary Key* – для усиления уникальности определения каждой строки в таблице.
- *Foreign Key* – для усиления заданного правила ссылочной целостности в случае, когда значение во внешнем ключе изменяется.
- *On Delete* – для усиления заданного режима ссылочной целостности в случае, если значение удаляется в поле первичного или внешнего ключа.
- *Unique (AK)* – для усиления правила ссылочной целостности, требующего, чтобы значения альтернативных ключей были уникальными.
- *Create/PK* – для включения системной процедуры, создающей первичный ключ в каждой таблице.
- *Create/FK* – для включения системной процедуры, создающей внешние ключи.

Trigger – это режимы триггера, которые позволяют переопределить шаблоны RI, устанавливаемые ERwin по умолчанию, с целью усиления ссылочной целостности. Возможные режимы следующие:

- *RI Type Override* – для переопределения шаблона, устанавливаемого по умолчанию, для всех связей, которые были присвоены определенному типу правила ссылочной целостности.
- *Relationship Override* – для переопределения шаблона, устанавливаемого по умолчанию, для какой-то конкретной связи.
- *Table* – режимы для таблиц позволяют указать, какие операторы языка определения данных будут использованы при создании схемы.

Возможны следующие режимы для таблицы:

- *CREATE TABLE* – для выполнения операторов *SQL CREATE TABLE* в процессе генерации схемы.
- *DROP TABLE* – для выполнения операторов *SQL DROP TABLE* перед выполнением операторов *CREATE TABLE* при генерации схемы.
- *Physical Storage* – для включения в схему объектов и параметров физической памяти.
- *Table Validation* – для включения операторов *SQL*, создающих правила, которые накладывают ограничения, для каждой сущности.
- *Pre-Script* – для включения в схему пре-скриптов (скриптов, выполняемых непосредственно перед генерацией схемы).
- *Post-Script* – для включения в схему пост-скриптов (скриптов, выполняемых непосредственно после генерации схемы).

Index – это режимы индексирования. Они указывают, каким образом будут создаваться и храниться индексы и какие из ключевых атрибутов будут индексированы. Возможные режимы следующие:

- *Primary Key (PK)* – для создания индекса по первичному ключу в каждой сущности.
- *Alternate Key (AK)* – для создания индекса по альтернативным ключам в каждой сущности.
- *Foreign Key (FK)* – для создания индекса по внешним ключам в каждой сущности.
- *Inversion Entry (IE)* – для создания индекса по инверсионным ключам в каждой сущности.
- *CLUSTERED* – для создания в схеме индекса *CLUSTERED*.
- *Physical Storage* – для включения в схему информации, относящейся к объектам физической памяти.

Column – режимы для колонок позволяют добавлять ограничения в операторы *SQL CREATE TABLE*. Выберите один или несколько возможных режимов:

- *Default* – для включения значения колонки по умолчанию в оператор схемы.
- *Physical Order* – для сохранения физического порядка расположения колонок при генерации новой схемы.
- *User Datatype* – для включения типа данных в оператор схемы, заданного пользователем для колонки.
- *Validation* – для включения в оператор схемы правило проверки введенных значений для колонки.

Other Options – другие доступные режимы. Они поддерживают специальные возможности, предоставляемые выбранной СУБД. Возможные режимы следующие:

- *Constraint Name* – для включения в схему имен ограничений.
- *Quote Names* – для заключения имен таблиц и колонок в кавычки.
- *Owner* – для включения владельца.

Примечание. При генерации схемы на сервере все изменения табличных характеристик, сделанные в Erwin, не распространяются на базу данных, если не будет удалена измененная таблица (*DROP*) и не будет создана заново (*CREATE*). Чтобы заменить старую таблицу на новую, поставьте метку в окна режимов *DROP TABLE* и *CREATE TABLE* в *Schema Generation*.

Установления соответствия между потоками данных и сущностями

Между BPwin и ERwin возможен обмен информацией о сущностях и их атрибутах для установления соответствия между информационными потоками (стрелками) в BPwin и сущностями в ERwin. Поскольку разработка сущностей может происходить как с помощью BPwin, так и ERwin, синхронизация моделей в процессе проектирования осуществляется с помощью механизма импорта/экспорта данных в файлы с расширением .eax (ERwin to BPwin) и .brx.

Экспорт сущностей и их атрибутов в ERwin осуществляется с помощью пункта меню *File/Export/BPwin*. При импорте (*File/Import/ERwin*) BPwin присваивает сущностям внутренние идентификаторы (*ID*), созданные в ERwin. Для

импорта сущностей и атрибутов без идентификаторов ERwin необходимо установить флажок *Import as new entities and attributes*, иначе будет установлена связь с исходной моделью в ERwin. Установка флажка *BPwin Only* означает использование сущностей и атрибутов только в модели BPwin. Флажок *Update by Name* означает, что идентификаторы ERwin будут игнорироваться. Идентификаторы необходимо сохранять для того, чтобы повторный импорт из ERwin в BPwin в связи с изменением сущности в ERwin сохранил связи сущности со стрелками. При установке флажка *Update by Name* связи будут созданы не по идентификаторам, а по именам сущностей и атрибутов. При импорте словарь сущностей и атрибутов будет автоматически пополнен.

Форматы .eas и .brx используются не только для добавления новых сущностей в модели ERwin или BPwin, но и для синхронизации процесса проектирования. С помощью этих файлов можно создать постоянную связь модели в BPwin и модели в ERwin на основе внутренних идентификаторов. При этом не важно, где (в BPwin или в ERwin) изначально создавались сущности.

Связь между моделями устанавливается с помощью следующих действий:

- экспорта сущностей и атрибутов в файл .brx. При этом BPwin автоматически ассоциирует файл .brx с моделью bp1;
- импорта файла .brx в модель ERwin. При этом BPwin обновляет содержание файла .brx, добавляя к сущностям внутренние идентификаторы ID, присвоенные в ERwin и исключает его повторный импорт в ERwin;
- импорта файла .brx в модель BPwin. В случае, если этот файл был открыт в ERwin, BPwin добавляет созданные идентификаторы в модель SADT.

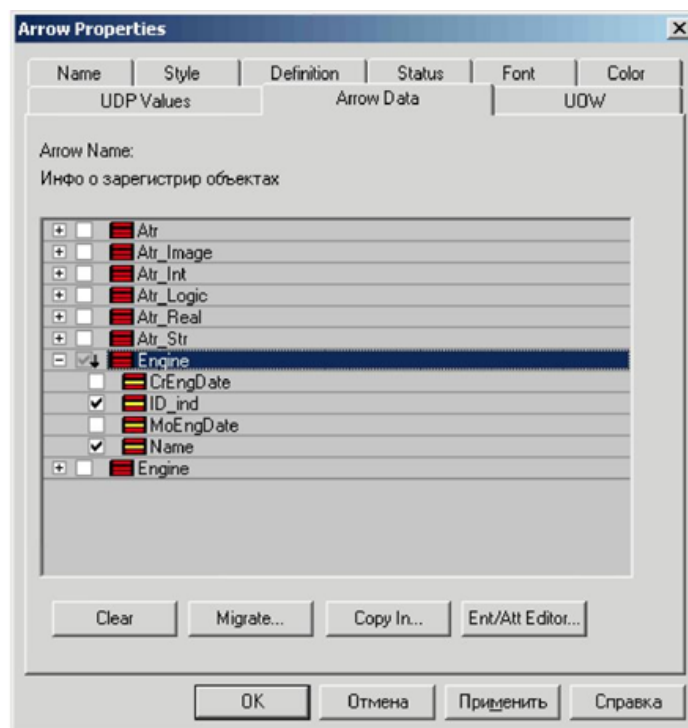


Рис. 10.1. Свойства стрелок

Установить соответствие между сущностями информационной модели и стрелками SADT модели автоматизированной информационной системы можно с помощью окна свойств стрелок (рис. 10.1).

Взаимодействие с PowerDesigner

В последних версиях PowerDesigner появилась возможность импортировать концептуальную и физическую модели, построенные в Erwin. Концептуальная модель импортируется как CDM (*Conceptual Data Model*), физическая – как PDM (*Physical Data Model*).

Для импорта файла в PowerDesigner выберите *File/Import/Erwin File* (CDM или PDM). Чтобы импортировать CDM и PDM, необходимо выбрать *File/Import/V6 Synchronized Models*.

Вы также можете создать в PowerDesigner новую концептуальную и физическую модели, для этого выберите *File/New/CDM* или *PDM*.

Принцип построения логической и физической модели в PowerDesigner не отличается от принципа построения Erwin.

При построении логической модели, в PowerDesigner используются инструменты:

- инструмент *Entity* – для создания сущности;
- инструмент *Relationship* – для создания связи между сущностями.

Чтобы сгенерировать физическую модель, необходимо выбрать *Tools -> Generate PDM*.

В свойствах сущностей можно указывать название (поля *Name* и *Code*), добавлять атрибуты (вкладка *Attributes*), идентификаторы (вкладка *Identifiers*), бизнес-правила (создаются отдельно *Model/Business Rules*). В свойствах связей можно указывать название связи (поля *Name* и *Code*), тип отношения сущностей (один-к-одному, один-ко-многим и др.), бизнес-правила и другие атрибуты.

Также есть возможность построения отчетов по созданным моделям (*Model/Reports/New* выбрать в поле *Report Template* необходимый тип отчета для CDM или PDM соответственно).

Экспериментальная часть занятия

1. С помощью индикатора типа модели переключитесь на физическую модель. Из меню *Tools* выберите *Forward Engineer/Schema Generation*. Появится диалог *SQL Server Scheme Generation* (рис. 10.2).

2. Щелкните кнопку *Preview*, расположенную внизу диалога, для предварительного просмотра. Появится *SQL Server Scheme Generation Preview* (рис. 10.3).

3. Для завершения просмотра схемы и возврата в диалог *SQL Server Scheme Generation*, нажмите кнопку *Close*. Для того чтобы изменить физическую модель данных для открытия в целевом сервере, нужно в меню базы данных (*Database*) выбрать *Choose Database*. Появится диалог Erwin для выбора целевого сервера – *Target Server* (целевой сервер).

4. Щелкните кнопку *Report*, которая расположена в нижней части диалога. В диалоге *Save As* в блоке *File Name* в качестве имени файла укажите «My ERwin Model.sql» и сохраните модель.

5. Из меню панели инструментов *Tools* выберите *Reverse Engineer*. Появится диалог *Reverse Engineer – Select Template*.

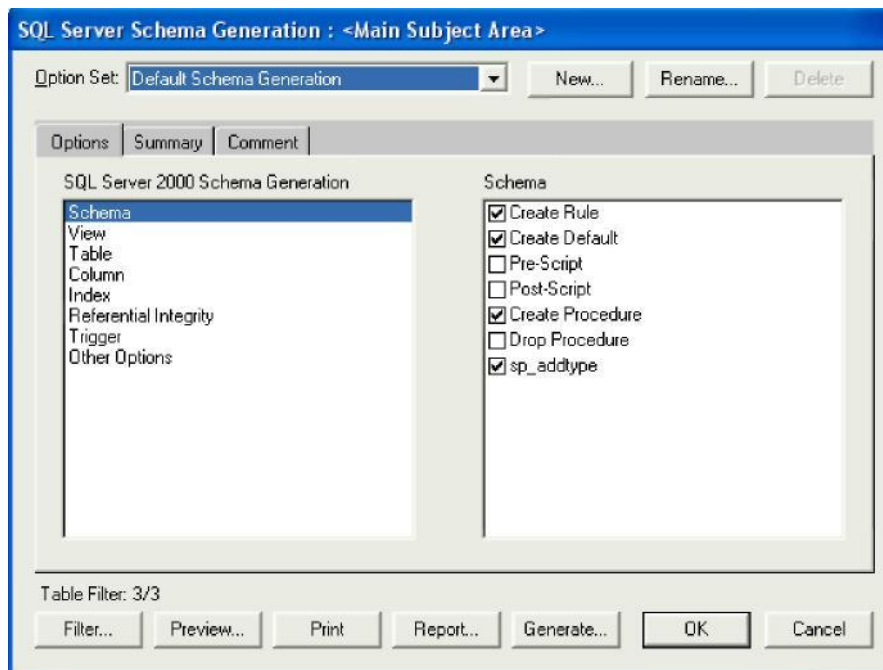


Рис.10.2. Схема генерации

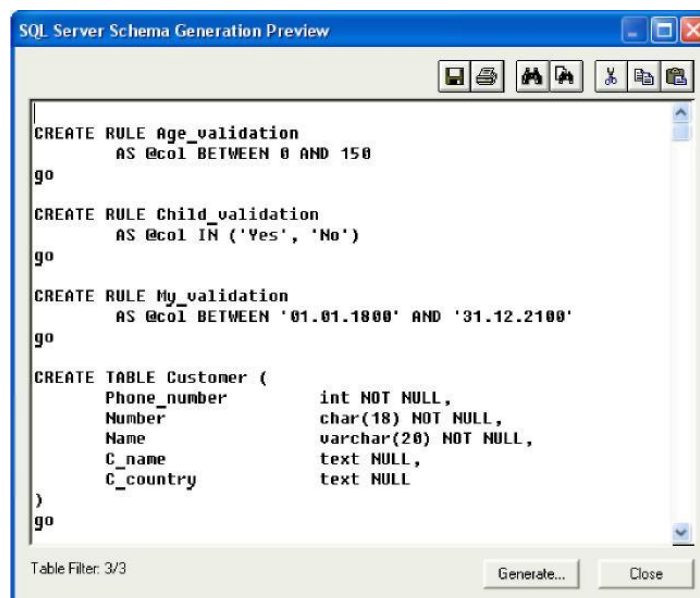


Рис. 10.3. Предварительный просмотр сгенерированной схемы

6. В диалоге выберите *Physical* в качестве нового типа модели (блок *New Model Type*). В качестве шаблона выберите *Blank Physical Model* и *SQL Server 2005/2008* в качестве целевой базы данных.

7. Щелкните *Next*. Появится диалоговое окно *Reverse Engineer Set Options* (установка опций обратного проектирования). В *Reverse Engineer From* выберите *Script File* и щелкните для просмотра *Browser*, чтобы расположить файл «My ERwin Model.sql», который был сохранен выше. Для данного задания следует принять по умолчанию установки в остальных областях диалога и продолжить операцию по кнопке *Next* (рис. 10.4). После проведенных операций будет виден небольшой диалог с текстом, который описывает структуру базу

данных. При завершении процесса обратного проектирования новая модель данных появляется в окне диаграммы (рис. 10.5).

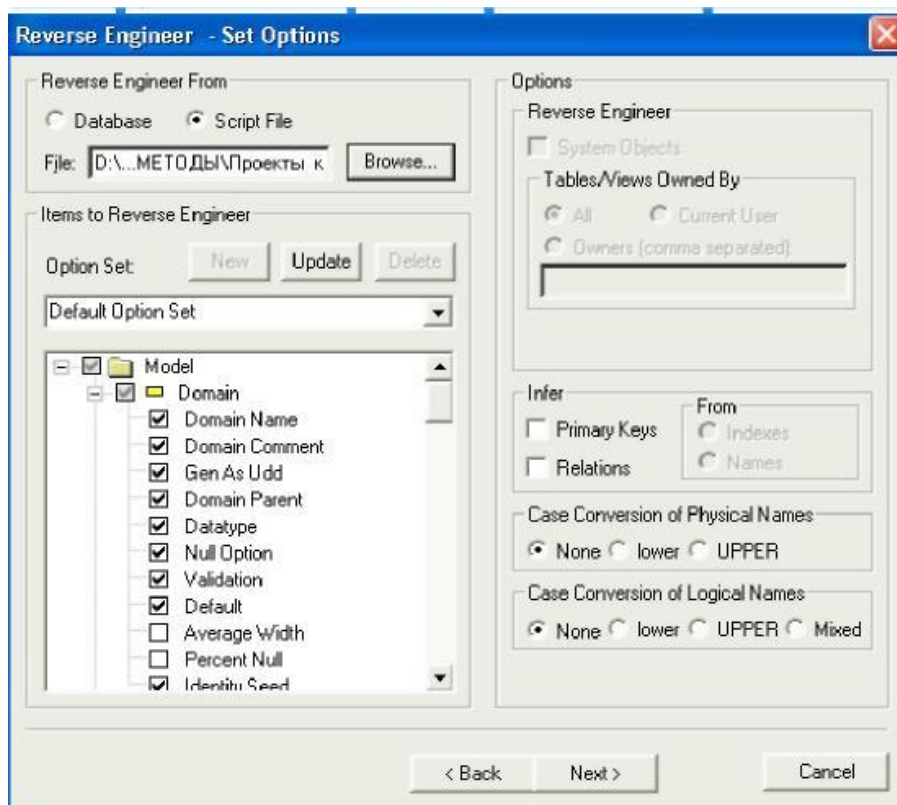


Рис. 10.4. Установка опций обратного проектирования

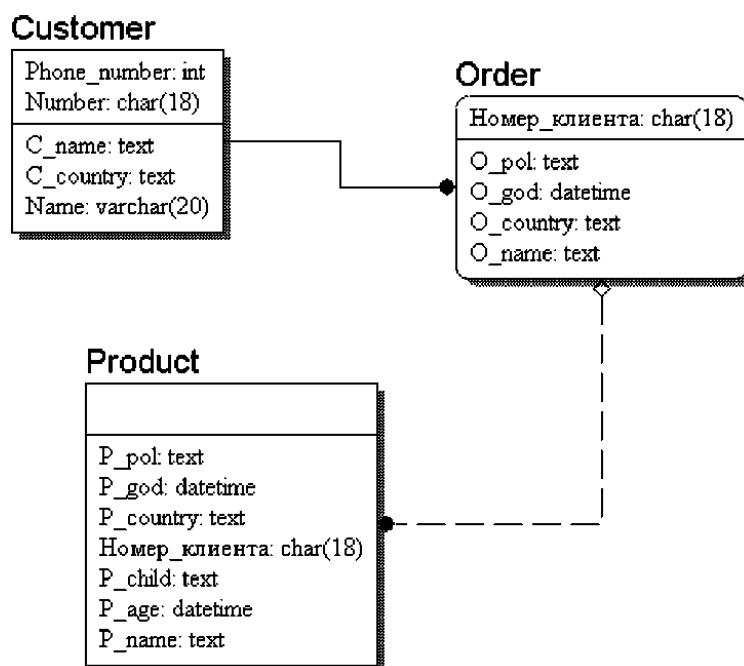


Рис. 10.5. Диаграмма в ERwin

8. В меню *File* выберите *Save* и сохраните модель как «lab_10.er1».

9. В *Model Explorer* в закладке *Model*, щелкните на плюс у фразы *Tables* для расширения списка объектов таблицы. Затем щелкните по плюсу у *CUSTOMER* для его расширения. Затем расширьте столбцы (*Columns*) под *CUSTOMER* и удалите столбец «*phone_number*». Сохраните модель.

10. В меню *Tools* выберите *Complete Compare*. Появится диалог *Complete Compare – Set Options*. В групповом блоке *Compare Type* выберите уровень сравнения объектов базы данных *Database Level Compare*:

- *Database level compare* – сравнение на уровне базы данных.
- *Model level compare* – сравнение на уровне модели.

Модель можно сравнивать с базой данных (*Database*), со сценарием файла (*Script File*), с файлом с расширением *er1*.

11. В групповом блоке *Compare Current Model With* выберите *Script File*, чтобы найти сценарий файла щелкните *Browse*, определите файл «*My ERwin Model.SQL*».

12. В групповом блоке *Sync Action* выберите *Update Current* и щелкните *Next*.

13. В диалоговом окне *Complete Compare – Items to Compare* примите все по умолчанию и щелкните *Next*.

14. В левой части окна можно отметить галочкой объекты, подлежащие синхронизации (или можно принять установки по умолчанию).

15. В диалоговом окне *Complete Compare – Object Filter Options* примите все по умолчанию (можно установить фильтр на объект установкой метки, что означает, что объект не будет сравнен) и щелкните *Next*. В этой точке ERwin начинает обрабатывать *Script File* (файл сценария).

16. В диалоговом окне *Complete Compare – Other Model Filter Options* примите все по умолчанию и щелкните *Next*.

17. Далее в диалоговом окне *Complete Compare – Resolve Differences* появятся два дерева возле друг друга. В них представлены различия в ERwin между текущей моделью и файлом сценария. Левое дерево помечено как «*lab_10*», а правое дерево помечено как «*My Erwin Model.sql*». На странице *Complete Compare – Resolve Differences* просмотрите различия между моделью и сценарием *SQL*. В правом дереве выберите столбец *Phone_Number* таблицы *CUSTOMER*, щелкнув по линии для его выделения.

18. Щелкните по кнопке *Import* в правой части диалога. Затем щелкните *Next*. Появится диалоговое окно *Complete Compare – Import Changes*. Нажмите кнопку *Start Import*. В окне появится сообщение *Done Importing*.

19. Для просмотра результатов необходимо нажать на кнопку *View Results*, в результате появится окно *Complete Compare – Import Summary*. Нажмите кнопку *Close* для закрытия окна просмотра.

20. Нажмите *Finish* для завершения импорта столбца *Phone_number* таблицы *CUSTOMER*.

21. В *Model Explorer* проверьте *phone_number*, который был повторно вставлен в модель после прежде удаленного атрибута.

22. Сохраните и закройте модель перед выходом.

Контрольные вопросы

1. Основная цель использования *ERWin*.
2. Какие бывают виды представления модели данных?
3. Что такое сущность?
4. Что такое атрибут?
5. Что такое связь?
6. Какие существуют виды связей?
7. Что такое мощность связи?
8. Что такое роль атрибута?
9. Что такое внешний ключ?

Практическое занятие № 11

ТЕМА: «ДИАГРАММА КЛАССОВ (CLASS DIAGRAM) И ДИАГРАММА ОБЪЕКТОВ (OBJECT DIAGRAM)»

Цель занятия: ознакомление с основными элементами проектирования и моделирования программных систем с помощью языка UML.

Задача: изучить методологию объектно-ориентированного проектирования на базе диаграммы классов и диаграммы объектов.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка диаграммы классов информационной системы.
3. Разработка диаграммы объектов информационной системы.

Теоретические основы занятия

Диаграмма классов (Class Diagram)

Диаграммы классов являются отправной точкой процесса разработки. Они помогают при анализе, позволяя аналитику общаться с клиентом в «привычных» ему терминах и стимулируя процесс выявления важных деталей в проблеме, которую требуется решить.

Класс – это категория или группа вещей, которая имеет сходные атрибуты и общие свойства. Атрибуты описывают перечень значений, в рамках которых указываются свойства объектов этого класса. Операция – это то, что может выполнять класс, либо то, что можно выполнять над данным классом.

В UML имена классов чаще всего включают несколько слов. Каждое слово в имени класса начинается с прописной буквы, пробелы между словами отсутствуют. Имена атрибутов и операций строятся по тем же правилам, но первая буква является строчной (например, «загрузитьПрограмму»).

Класс представляется прямоугольником, разделенным на три части:

- верхняя часть содержит имя класса;
- средняя часть содержит атрибуты;
- нижняя часть содержит операции.

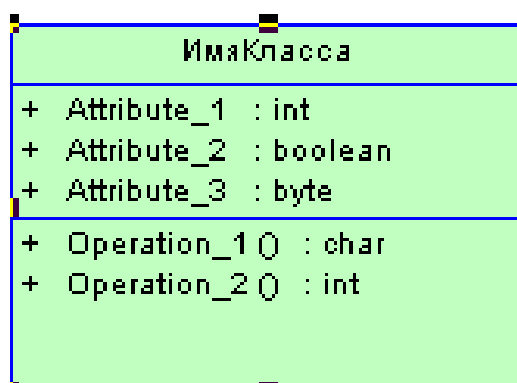


Рис. 11.1. Пример класса

Рассмотрим основные элементы приложения PowerDesigner при создании диаграммы классов.

Создание класса. Оно выполняется с помощью инструмента *Class* на панели инструментов *Pallette*. В свойствах (*Class Properties*) указываются:

- вкладка *General* – название класса и название в программном коде в полях *Name* и *Code*;
- вкладка *Attributes* – вносятся атрибуты класса. Каждый атрибут имеет свои свойства, их можно указать в свойствах атрибута (для этого следует нажать на введенный атрибут два раза левой кнопкой мыши – откроется форма *Attribute Properties*). В свойствах атрибута заполняются поля *Name* и *Code*, выбирается тип для каждого значения атрибута (строка *Data Type*). Для того чтобы указать значение атрибута по умолчанию, необходимо перейти на вкладку *Detail* и в поле *Initial value* вписать заданное значение;
- вкладка *Operations* – вносятся операции, которые может выполнять данный класс. Каждая операция имеет свои свойства, их можно указать в свойствах операции (для этого нажать на введенный атрибут два раза левой кнопкой мыши – откроется форма *Operation Properties*). В свойствах операции заполняются поля *Name* и *Code*, выбирается тип возвращаемого значения (строка *Return Type*). Также возможно указать параметры операции и их типы (вкладка *Parameters*).

Если список атрибутов и/или операций слишком велик, то можно уточнить информацию с помощью стереотипа (поле *Stereotype*).

Также дополнительную информацию можно внести в поле комментария *Comment* или присоединить комментарий к классу с помощью инструмента *Note*.

Ассоциации. Если классы взаимодействуют друг с другом, то такое взаимодействие называется ассоциацией и изображается в PowerDesigner при помощи инструмента *Association*. В свойствах *Association Properties* указывается (вкладка *General*) название ассоциации и название в программном коде (поля *Name* и *Code*).

Если один класс ассоциируется с другим, каждый из них играет свою роль в этой ассоциации. Роли указываются в полях *Role A* (для класса A) и *Role B* (для класса B).

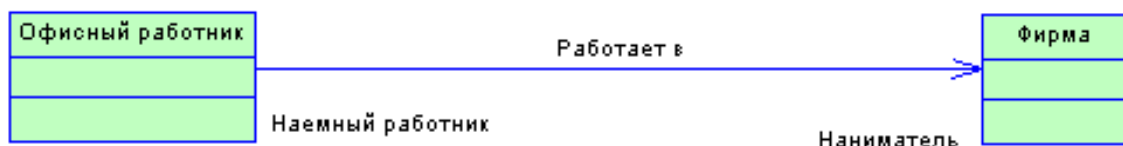


Рис. 11.2. Ассоциация роли между классами

Ассоциации могут работать в разных направлениях, а также могут быть более сложными – с одним классом могут ассоциироваться несколько других.

Подобно классам ассоциация может иметь атрибуты и операции. Для отображения класса ассоциации в PowerDesigner используется тот же

инструмент *Association*, но соединяются не два класса, а ассоциация и класс. А класс ассоциации сам может быть связан с другими классами (рис. 11.2).

Кратность. Это – количество объектов одного класса, которые могут быть связаны с определенным количеством объектов другого класса (рис. 11.3). В PowerDesigner следует открыть в свойствах ассоциации вкладку *Detail* и установить кратность в поле *Multiplicity*. Кратность может быть один-к-одному, один-ко-многим, многие-к-одному, многие-ко-многим.

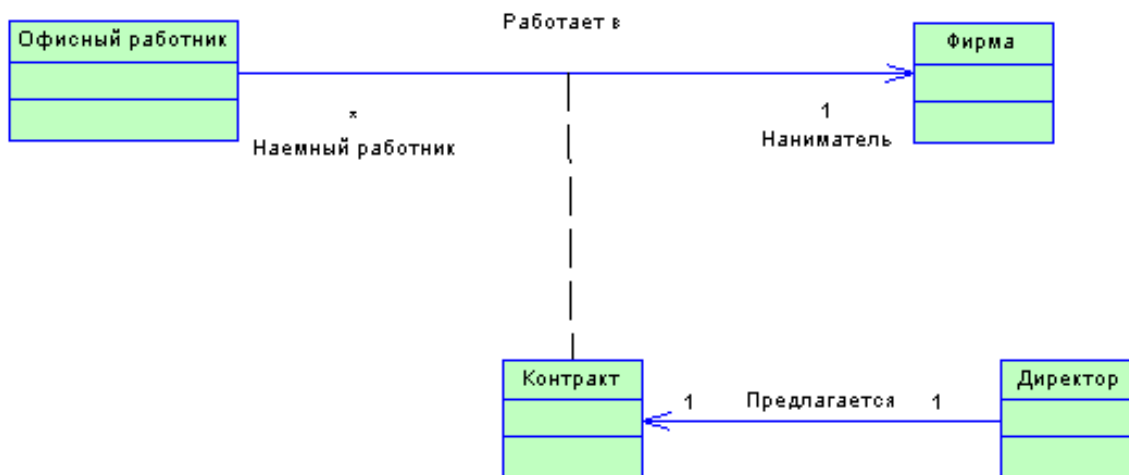


Рис. 11.3. Отображение кратности между классами

Класс может ассоциироваться сам с собой (рефлекторная ассоциация). Пример представлен на рис. 11.4.

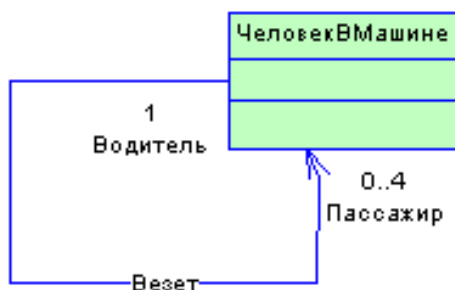


Рис. 11.4. Рефлекторная связь класса

Наследование. Класс может наследовать атрибуты и операции другого класса. Наследующий класс является дочерним по отношению к родительскому, от которого он наследуется. Связь родительского и дочернего классов соответствует отношению «является видом». В PowerDesigner для изображения наследования используется инструмент *Generalization*.

Абстрактные классы предназначены только для использования в качестве базовых для наследования и не порождают своих объектов.

Зависимость. Взаимосвязь при использовании одного класса другим называется зависимостью. Наиболее общий случай зависимости – это использование одного класса в сигнатуре операции другого класса. Для изображения зависимости применяется инструмент *Dependency*. На рис. 11.5 изображены два класса: «Система» и «Форма». У класса «Система» есть

операция «отобразитьФорму()». Отображаемая системой форма зависит от того, какой экземпляр класса «Форма» выбрал пользователь.

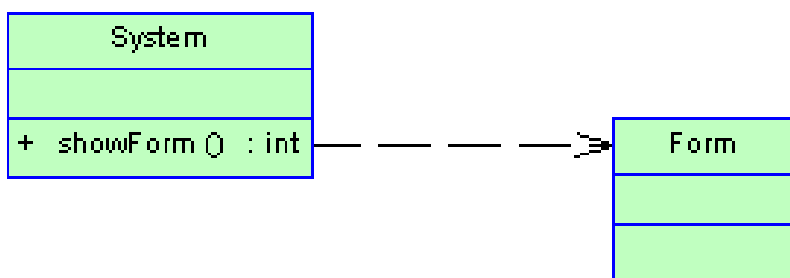


Рис. 11.5. Зависимость между классами

Агрегация. Взаимосвязь, при которой класс состоит из некоторого количества классов-компонентов, называется агрегацией. Компоненты и класс, который они составляют, находятся в ассоциации «часть-целое». Агрегацию можно представить в виде дерева, корнем которого является «целое», а листьями – его компоненты. В PowerDesigner применяется инструмент *Aggregation*. Чтобы проиллюстрировать агрегацию на примере, смоделируем диаграмму «Меню в ресторане» (рис. 11.6). В свойствах каждого класса и ассоциации необходимо указать все те поля, описание которых приведено выше.

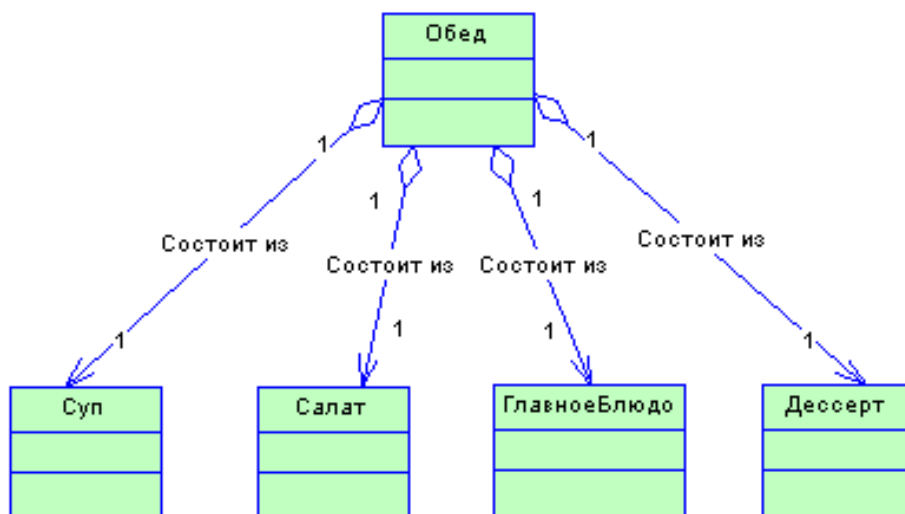


Рис. 11.6. Агрегация класса «Обед»

Композит. Это – строгий тип агрегации, характеризующийся тем, что каждый элемент может принадлежать только одному целому. Например, компоненты стола – столешница и ножки – составляют композит. Для отображения в PowerDesigner используется инструмент *Composition*.

Интерфейсы и реализация. Интерфейс – это набор операций, которые задают некоторые аспекты поведения класса и представляют его для других классов.

Диаграмма объектов (Object Diagram)

Объект – это экземпляр класса – особая сущность, которая имеет заданные значения атрибутов и операций. В UML объект изображается также как и класс в виде прямоугольника, но с подчеркнутым именем. Наименование экземпляра размещено слева от двоеточия, а наименование класса – с правой стороны. Имя объекта начинается со строчной буквы. Существуют анонимные объекты (объект принадлежит какому-то классу, но не имеет имени).

Диаграмма объектов содержит конкретную информацию об экземплярах класса и их связях с другими экземплярами во времени.

Экспериментальная часть занятия

PowerDesigner состоит из: окна просмотра, рабочей области, панели инструментов и окна для вывода данных (рис. 11.7). Вы можете настроить рабочее пространство так, как вам будет удобно.

Чтобы создать новый проект, необходимо открыть: *File/New*, выбрать необходимую диаграмму. В данном случае будем рассматривать объектно-ориентированную модель (*Object-Oriented Model*). Аналогично создаются бизнес-модель, концептуальная и физическая модели, а также возможно сформировать отчеты. При создании объектно-ориентированной модели, необходимо указать язык программирования и модель диаграммы, которая будет создана первой.

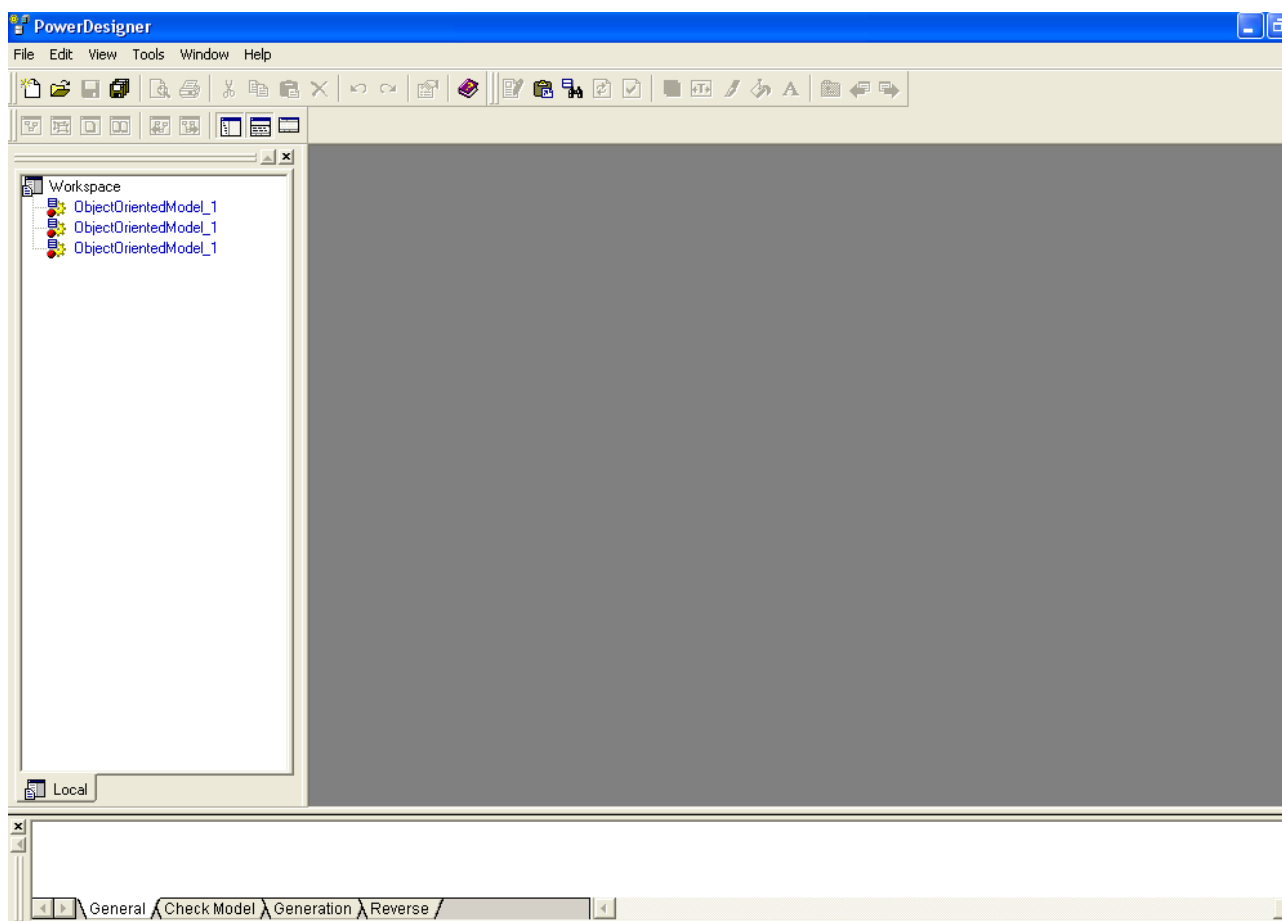


Рис. 11.7. Рабочая область PowerDesigner

Чтобы создать новую диаграмму в существующем проекте, необходимо в рабочей области нажать на правую кнопку мыши и выбрать *Diagram/New Diagram* и необходимый тип диаграммы (*Class Diagram*, *Use-Case Diagram*).

Создание диаграммы классов

1. Необходимо создать диаграмму классов: *Diagram/New Diagram/Class Diagram*.

Чтобы нарисовать саму диаграмму, необходимо переносить выбранные на панели инструментов объекты на рабочую область.

Чтобы открыть свойства созданного с помощью инструмента объекта, необходимо нажать на него два раза левой кнопкой мыши.

Чтобы в созданной диаграмме были видны все объект и связи, необходимо настроить видимость данных объектов на рабочей области, для этого необходимо открыть *Tools/Display Preferences* и поставить галочки рядом с необходимыми полями.

2. Для того чтобы на диаграмму классов добавить интерфейс, необходимо:

- *при первом способе*: добавить его с помощью инструмента *Interface*. Свойства заполняются аналогично свойствам класса;

- *при втором способе*: добавить его автоматически, нажав правую кнопку мыши на созданном классе, и в списке выбрать *Create Interface*. В этом случае операции, прописанные в созданном классе, добавятся автоматически.

Взаимосвязь между классом и его интерфейсом называется отношением *реализации* (инструмент *Realization*).

Рассмотрим пример (рис. 11.8).

1. При использовании автомата по продаже карт экспресс оплаты, мы взаимодействуем с ним через интерфейс «ПанельУправления».

2. Отношение между интерфейсом «ПанельУправления» и объектом «АвтоматПоПродажеКартОплаты» – это *реализация*.

3. Отношение между объектом «Человек» и интерфейсом «ПанельУправления» – это *зависимость* (инструмент *Dependency*).

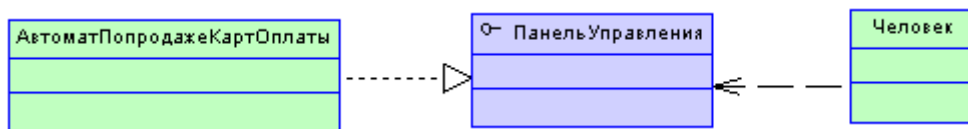


Рис. 11.8. Пример диаграммы классов

Видимость. Термин «видимость» применяется по отношению к атрибутам и операциям.

Выделяют три области видимости:

- открытую (могут использовать другие классы);
- защищенную (могут использовать только наследники данного класса);
- закрытую (используются только самими классами).

В PowerDesigner видимость указывается в свойствах во вкладке *General*, в поле *Visibility*.

Создание диаграммы объектов

1. Создайте диаграмму объектов с помощью меню **Diagram/New Diagram/Object Diagram**.
2. Создайте объект с помощью инструмента *Object*. В свойствах каждого объекта задайте название объекта в полях *Name* и *Code*.
3. В поле *Class* выберите из выпадающего списка класс, к которому относится объект.
4. В результате на вкладке *Attribute Value* можно увидеть атрибуты класса. Если поставить галочку рядом с атрибутом класса, то он будет являться атрибутом объекта.
5. Соединить объекты с помощью инструмента *Instance Link*.
6. Зависимость можно показать с помощью инструмента *Dependecy*. Результат диаграммы объектов представлен на рис. 11.9.

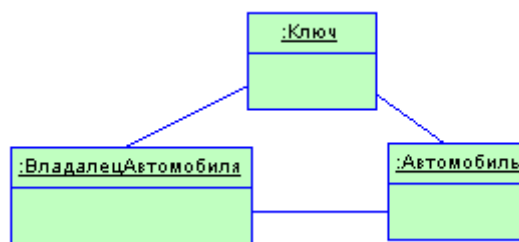


Рис. 11.9. Пример диаграммы объектов

Контрольные вопросы

1. Назовите и охарактеризуйте виды отношений между объектами.
2. Назовите и охарактеризуйте виды отношений между классами.
3. Перечислите типы мощности ассоциации.

Практическое занятие № 12

ТЕМА: «ДИАГРАММЫ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ (USE-CASE DIAGRAM)»

Цель занятия: изучить навыки создания диаграммы вариантов использования.

Задача: изучить навыки создания диаграммы вариантов использования.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка диаграммы вариантов использования системы.

Теоретические основы занятия

Перед приобретением какого-либо продукта или системы, мы формируем список требований, которые желаем видеть в нашем продукте, например возможности будущей системы, функции, которые она будет выполнять и т.д., т.е. анализируем прецеденты (*use-case analysis*). Такой процесс играет особенно важную роль на этапе анализа системных требований. Способ использования системы пользователями определяет проектное решение, которое будет положено в ее основу.

Прецедент – это конструкция, позволяющая описать систему с точки зрения потенциальных пользователей.

Прецедент представляет собой набор сценариев использования системы. Каждый сценарий описывает последовательность действий. Каждая последовательность действий инициируется пользователем, другой системой, аппаратным средством или в какой-либо момент времени. Сущности, инициирующие сценарии, называются исполнителями (*actor*).

Результат прецедента должен быть полезен исполнителю, инициировавшему этот прецедент, либо какому-то другому исполнителю.

Прецедент заставляет потенциальных пользователей думать о системе со своих позиций. Пользователям не всегда легко выразить свои требования к системе. Основная идея – привлечь пользователей к разработке системы на ранних стадиях анализа и проектирования. Это повышает вероятность создания полезной системы, позволяет сконцентрироваться на мнении людей, а не на компьютерных понятиях, с которыми не могут работать реальные пользователи.

Лучший способ определения прецедентов – это опрос. При этом важно выявить предусловия для инициализации прецедента и постусловия, реализуемые в результате выполнения прецедента.

Для аналитиков прецедент помогает определить способ использования системы. Для разработчиков – это полезный инструмент, предоставляющий надежную методику формирования требований к системе с точки зрения пользователя (заказчика).

Представление модели прецедентов следующее:

- один исполнитель инициирует прецедент;
- другой исполнитель получает новое качество от его реализации.

Графически модель прецедентов представляется просто: эллипс соответствует прецеденту, а упрощенная фигурка представляет исполнителя. Иницирующий исполнитель находится слева от прецедента, принимающий исполнитель – справа. Взаимодействие исполнителя и прецедента обозначается с помощью сплошной соединительной линии.

При выполнении анализа прецедентов определяются границы системы и ее связь с окружающим миром. Исполнители обычно находятся вне системы, а прецеденты – внутри. Для обозначения границ системы используется прямоугольник, внутри которого указывается имя системы.

Исполнители, прецеденты и соединительные линии образуют модель прецедентов.

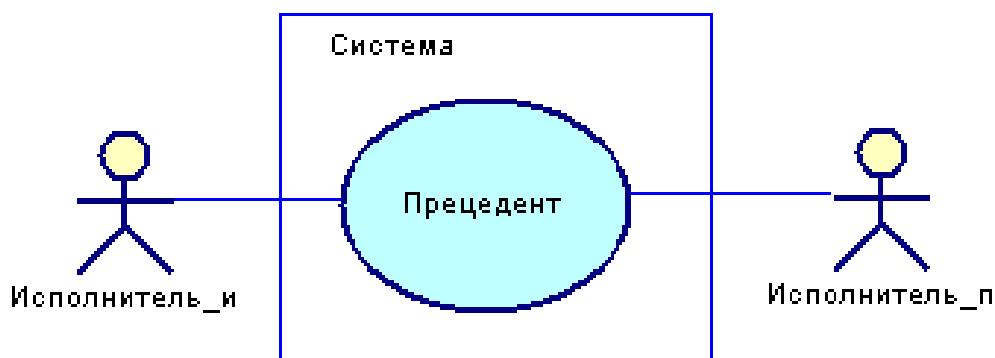


Рис. 12.1. Пример диаграммы вариантов использования

Экспериментальная часть занятия

Рассмотрим построение прецедентов на примере модели автомата по продаже карт экспресс-оплаты.

1. Создайте диаграмму *Use-Case*.

2. Далее создайте три прецедента, используя инструмент *Use-Case*. Чтобы изменить имя прецедента и другие свойства, нажмите дважды левой кнопкой мыши на объект. В открывшемся окне в поле *Name* введите «Покупка карты», а в поле *Code* введите название объекта, которое затем будет использоваться в программном коде «BuyCard». Остальные *use-case* переименуйте в соответствии с табл. 12.1.

Таблица 12.1

Свойства прецедентов

Прецедент	Name	Code
Case_1	Покупка карты	BuyCard
Case_2	Заправка автомата	Refuelling
Case_3	Сбор денег	GetMoney

3. Создайте шесть исполнителей, нажав на кнопку *Actor*. Переименуйте объекты в соответствии с табл. 12.2.

Свойства исполнителей

Исполнитель	Name	Code
Actor_1	Покупатель	Customer
Actor_2	Покупатель	Customer
Actor_3	Специалист по заправке	Refueller
Actor_4	Специалист по заправке	Refueller
Actor_5	Инкассатор	Collector
Actor_6	Инкассатор	Collector

4. Создайте взаимосвязи между объектами. Взаимосвязи могут идти в двух направлениях:

- от исполнителя к прецеденту (связь *Primary actor*);
- от прецедента к исполнителю (связь *Secondary actor*).

Создайте взаимосвязи от иницирующих исполнителей к прецедентам и от прецедентов к принимающим исполнителям с помощью инструмента *Assosiation*. Для этого подведите указатель к исполнителю, нажмите левую клавишу мыши и, удерживая ее, подведите к прецеденту. Соедините объекты так, как показано на рис. 12.2.

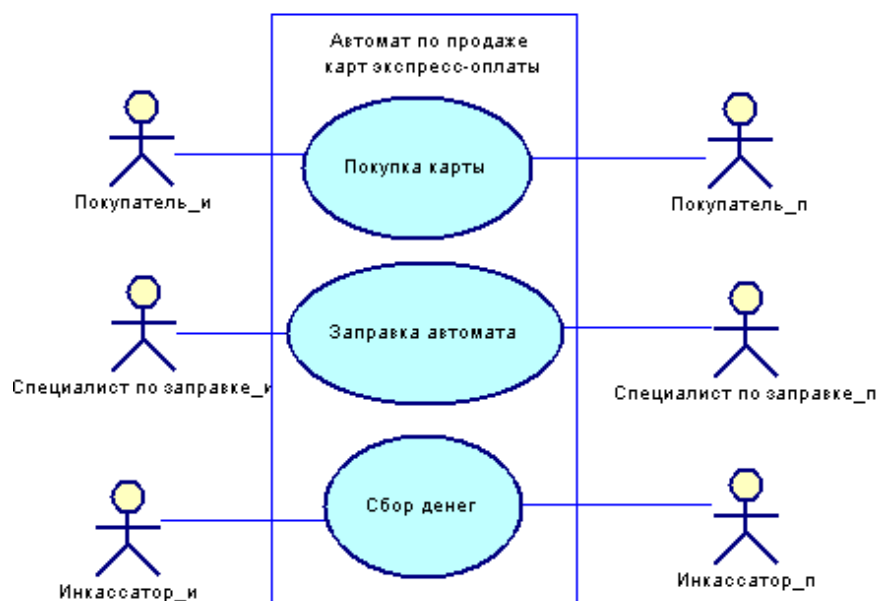


Рис. 12.2. Взаимосвязи между прецедентами и исполнителями

5. Чтобы повторно использовать шаги одного прецедента в другом, применяется *включение*. Рассмотрим прецедент «Заправка автомата» и «Сбор денег». Они оба начинаются с разблокирования и открытия автомата и заканчиваются закрытием и блокировкой. Прецедент «Проникновение внутрь» включает первые два шага, а «Выход наружу» – два остальных. Прецеденты «Заправка автомата» и «Сбор денег» включают в себя прецеденты «Проникновение внутрь» и «Выход наружу». Графически включение обозначается в виде соединительной пунктирной линии со стрелкой, указывающей на тот класс, от которого зависит другой (инструмент *Dependency*). В свойствах данного объекта необходимо в поле *Stereotype*

указать слово <<включает>>. Результирующая модель прецедентов с включением представлена на рис. 12.3.

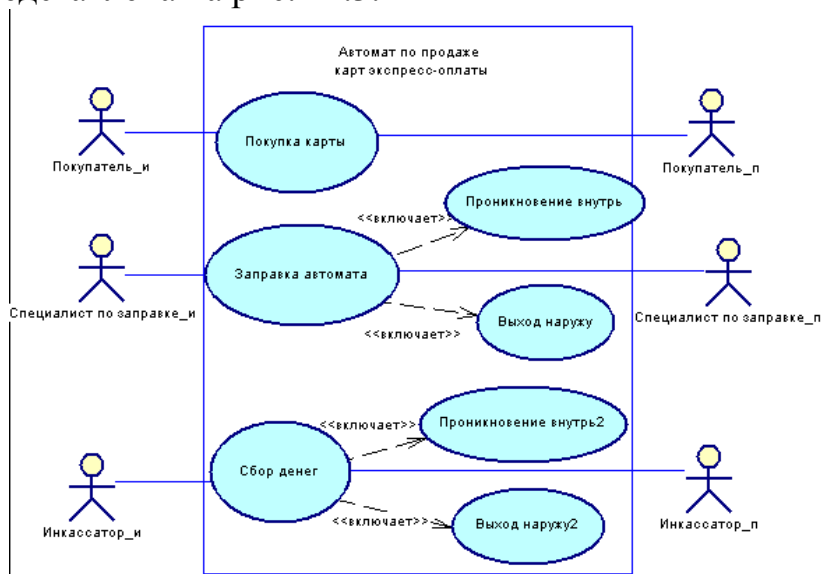


Рис. 12.3. Диаграмма вариантов использования с включением

6. Можно расширить исходный (базовый) прецедент за счет добавления новых шагов. Данный тип взаимоотношений называется *расширением*. Расширение может происходить только на заданных точках последовательности шагов базового прецедента. Такие места называются точками расширения. Подобно включению, расширение отображается линией зависимости (пунктир со стрелкой) со стереотипом <<расширяет>> (инструмент *Dependency*).

7. В соответствии с рис. 12.4 расширим прецедент «Заправка автомата»: в отличие от равномерного пополнения запасов всех сортов сиропа этот прецедент позволяет пополнять автомат с учетом спроса.

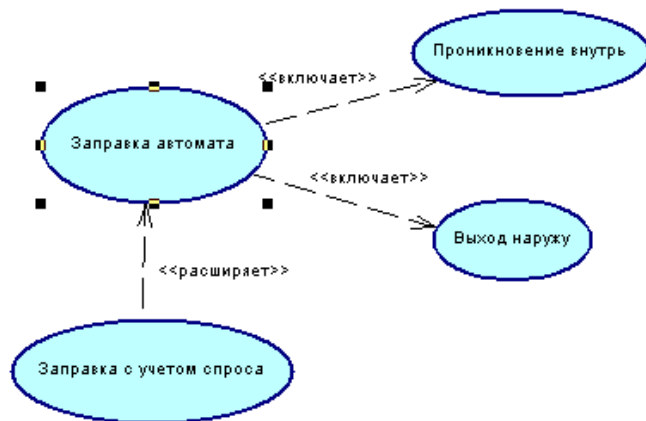


Рис. 12.4. Расширение диаграммы вариантов использования

Контрольные вопросы

1. Назовите основные элементы диаграммы вариантов использования.
2. Что в диаграммах вариантов использования называется актером?
3. Поясните применение отношения ассоциации на диаграмме.

Практическое занятие № 13

ТЕМА: «ДИАГРАММЫ СОСТОЯНИЙ (STATECHART DIAGRAM)»

Цель занятия: изучить навыки создания диаграммы состояний.

Задача: изучить навыки создания диаграммы состояний.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка диаграммы состояний информационной системы.

Теоретические основы занятия

Рассмотрим новую категорию элементов, которая позволяет описать поведение системы и показать, как части модели UML изменяются во времени. К одной из таких категорий относятся элементы диаграммы состояний.

Изменение в системе можно охарактеризовать так: объекты изменяют свое состояние в ответ на происходящие события и с течением времени. Например, при щелчке на кнопке пульта дистанционного управления телевизор изменяет свое состояние и показывает передачи другого канала и т.д. Поэтому диаграмма состояний, охватывая приведенный выше тип изменения, представляет состояния объекта и переходы между ними, а также показывает начальное и конечное состояние объекта. Данная диаграмма значительно отличается от диаграммы классов, диаграммы объектов или диаграммы прецедентов. Диаграмма состояний показывает состояния одного объекта.

Состояние изображается прямоугольником со скругленными углами, переход – сплошной линией со стрелкой. Закрашенный круг соответствует начальной точке последовательности состояний, а обведенный круг («глазок») представляет конечную точку (рис. 13.1).

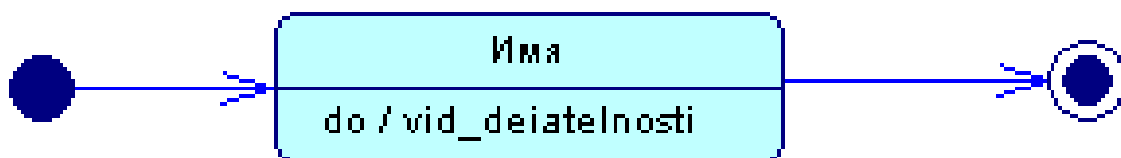


Рис. 13.1. Пример состояния

При этом прямоугольник разделен на две части. Верхняя часть содержит имя состояния, которое нужно присвоить независимо от присутствия других элементов, а нижняя предназначена для размещения видов деятельности.

Также необходимо уточнить, что переход зачастую происходит в ответ на переключающее событие и может вызывать выполнение некоторых действий.

Например, можно указать событие, которое привело к переходу (переключающее событие) и выполняемые вычисления (действия), которые приводят к изменению состояния.

Иногда событие вызывает переход баз всякого действия, иногда же переход происходит из-за того, что в текущем состоянии выполнены все действия (не из-за события). Такой тип перехода называется безусловным переходом.

На диаграмме состояний отображаются все переходы между состояниями одного объекта системы. Если информация слишком детализирована, то диаграмма может слишком усложниться.

Диаграммы состояний используются аналитиками, проектировщиками и разработчиками для исследования поведения объектов в системе. Диаграмма классов и диаграмма объектов отображают только статическое состояние системы. На них представлены иерархии и ассоциации, и из них можно узнать о возможном перечне действий системы, но ничего нельзя узнать о деталях динамического поведения.

Однако разработчики должны понимать поведение объектов, поскольку их задачей является реализация этого поведения в программном обеспечении. Просто разработать объект недостаточно: специалисты должны добиваться того, чтобы объект выполнял свои функции. Диаграммы состояний дают полную информацию о желаемом поведении. Ясное представление о поведении объекта повышает вероятность того, что группа разработчиков создаст систему, удовлетворяющую выдвинутым требованиям.

Экспериментальная часть занятия

Рассмотрим построение диаграммы состояний в PowerDesigner на примере графического пользовательского интерфейса компьютера.

Предположим, что упрощенный интерфейс может находиться в одном из трех состояний:

- «Инициализация»;
- «Работа»;
- «Завершение работы».

При включении компьютера происходит загрузка. Поэтому включение компьютера является переключающим событием, которое приводит к переходу интерфейса в состояние «Инициализация», а загрузка – это действие, происходящее во время перехода. Результатом выполнения действия в состоянии «Инициализации» является выработка переключающего события, которое вызывает переход в состояние «Работа». При щелчке на кнопке завершения работы, осуществляется переход в состояние «Завершение работы» и компьютер выключается.

1. Создайте новую диаграмму. Для этого следует нажать правую кнопку мыши в пустом месте рабочей области и выбрать пункт контекстного меню *Diagram/New Diagram)/ StateChart Diagram*.


2. Введите название диаграммы (поле *Name*) «Диаграмма состояний GUI», а в поле *Code* укажите название диаграммы понятное для разработчиков при реализации проекта на определенном языке программирования, например «StateChart_Digram_GUI». Чтобы название в поле *Code* не дублировало значение поля *Name*, снимите выделение кнопки «=» справа от поля *Code*.


3. Создайте три объекта с помощью инструмента *State*, который расположен на панели инструментов *Pallete*, а также начальное (*Start*) и конечное состояние (*End*). В свойствах переименуйте все объекты состояний в соответствии с табл. 13.1.

Таблица 13.1

Свойства объектов состояний

Объекты состояний	Name	Code
State_1	Инициализация	Initialization
State_2	Работа	Work
State_3	Завершение работы	EndOfWork

4. Далее необходимо добавить состояния и переходы. Для этого откройте свойства объекта «Инициализация» и во вкладке действия (*Actions*) с помощью кнопки *Add a row* , добавьте новое действие. В столбце *Trigger Event* выберите состояние выполнение (*do*), в столбце *Name* введите название действия *Loading*. Нажмите кнопку *Применить* для сохранения внесенных изменений.

5. Соедините последовательно все объекты с помощью инструмента *Transition* и дайте названия переходам, как показано на рис. 13.2. Для этого зайдите в свойства объекта *Transition* и во вкладке *Trigger* в поле *Trigger Event* создайте новое действие (кнопка *Create* ).

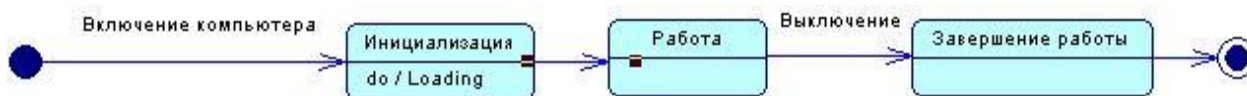


Рис. 13.2. Результат выполнения пункта 5

Переход может происходить в случае специальных условий – так называемых условий перехода. Рассмотрим и дополним пример, приведенный выше. Если на компьютере в течение 15 мин не выполняется никаких действий, то активизируется экранная заставка. В результате пользовательский интерфейс переходит из состояния «Работа» в состояние «Отображение заставки». 15-минутный интервал в данном случае является условием перехода.

6. Дополним нарисованную выше диаграмму, создадим дополнительное состояние с названием «Отображение заставки» и два перехода. Переход от состояния «Отображение заставки» к состоянию «Работа» назовем «Нажатие клавиши или перемещение указателя». Для перехода от состояния «Работа» к состоянию «Отображение заставки» зададим условие, для этого откроем окно со свойствами перехода, перейдем на вкладку условия (*Condition*) и в поле *Alias* введем условие «Время истекло». Условие отображается на диаграмме в квадратных скобках. После сохранения свойств результат должен соответствовать рис. 13.3.

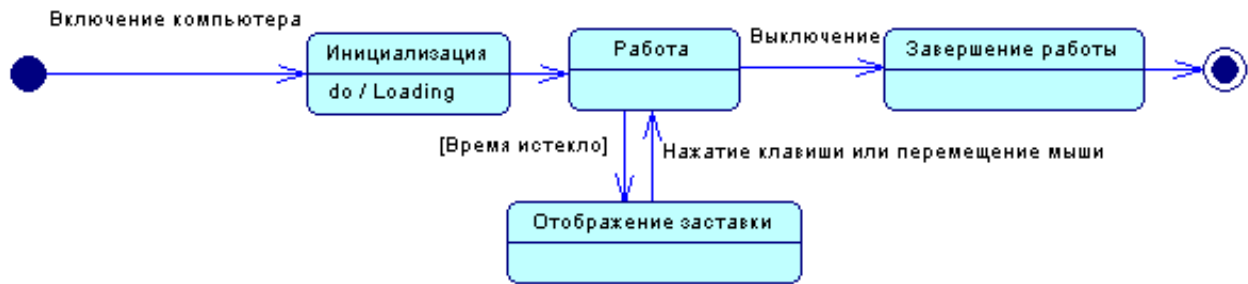


Рис. 13.3. Результат выполнения работы

Контрольные вопросы

1. Назовите назначение диаграммы состояний.
2. Назовите основные элементы диаграммы состояний.
3. Каким образом формируются условия перехода?

Практическое занятие № 14

ТЕМА: «ДИАГРАММЫ ПОСЛЕДОВАТЕЛЬНОСТЕЙ (SEQUENCE DIAGRAM)»

Цель занятия: изучить навыки создания диаграммы последовательностей.

Задачи:

1. Изучить теоретический материал.
2. Изучить навыки создания диаграммы последовательностей.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка диаграммы последовательностей информационной системы.

Теоретические основы занятия

Если необходимо показать взаимодействие объектов друг с другом, то без указания времени не обойтись. Основная идея в том, что взаимодействие объектов происходит в заданной последовательности, и для выполнения этой последовательности от начала до конца требуется время. Такая последовательность задается в процессе разработки системы, и для ее отображения используется диаграмма последовательности UML.

Диаграмма последовательностей состоит из обычных объектов, представленных в виде прямоугольников (с подчеркнутыми именами), сообщений, изображенных сплошными линиями со стрелками, а также вертикальной оси времени, определяющей последовательность событий.

Объекты располагаются в верхней части диаграммы слева направо. Под каждым объектом расположена пунктирная вертикальная линия, которая называется линией жизни этого объекта. Вдоль линии жизни располагаются узкие длинные прямоугольники, которые называются точками активации. Точка активации представляет выполнение объектом некоторой операции. Длина прямоугольника соответствует длительности процесса активации.

Сообщения, передаваемые от одного объекта к другому, на диаграмме изображаются в виде линий, соединяющих линии жизни этих объектов. Объект может передать сообщение самому себе.

Существует два типа сообщений:

- вызов – запрос объекта-отправителя к объекту-получателю на выполнение одной из его операций. Обычно отправитель ждет завершения выполнения операции. Такой тип сообщений называется *синхронным*, так как в этом случае отправитель «синхронизирует» свои действия с получателем. Если сообщение *асинхронное*, то отправитель передает управление получателю и не ожидает ответа для продолжения выполнения своих действий;
- ответ – ответ объекта-получателя на вызов объекту-отправителю сообщения.

Время на диаграмме изменяется вдоль вертикального направления. Начало отсчета находится вверху, увеличение времени происходит сверху вниз. Чем позже передается сообщение, тем ближе к нижней части диаграммы последовательностей оно располагается.

Основной набор обозначений приведен на рис. 14.1.

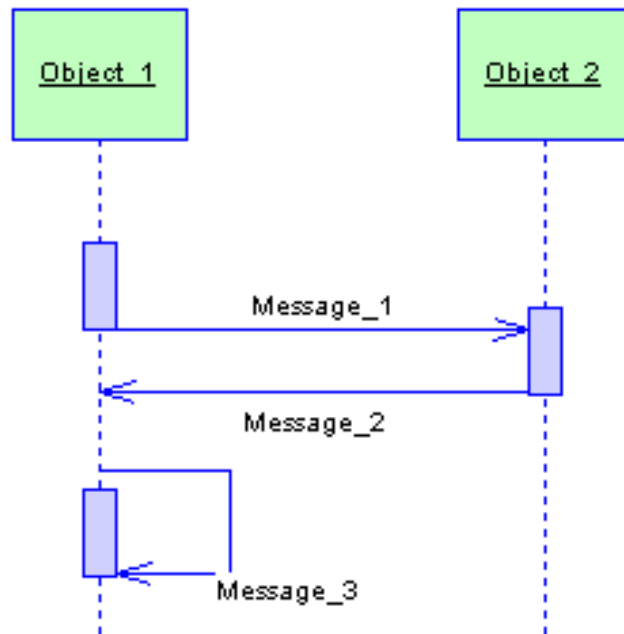


Рис. 14.1. Пример диаграммы последовательностей

Экспериментальная часть занятия

Построение диаграммы последовательностей в PowerDesigner рассмотрим на упрощенном примере практического занятия №12 (автомат по продаже карт экспресс-оплаты).

Предположим, что в реализации данного процесса участвуют три объекта:

- лицевая панель;
- реестр для сбора денег;
- отсек для хранения карт оплаты.

Функции лицевой панели:

- принимать деньги и заказ;
- отображать приглашения;
- получать сдачу от реестра и выдавать ее покупателю;
- выдавать сдачу;
- получать карту оплаты из отсека, где она хранится, и передавать ее покупателю.

Функции реестра:

- получать заказ покупателя от лицевой панели;
- осуществлять подсчет денег;
- рассчитывать сдачу.

Функции отсека для хранения карт оплаты:

- проверять наличие выбранного типа карты оплаты;

- выдавать карту оплаты.

Выполним модификацию диаграммы классов в соответствии с рис. 14.2.

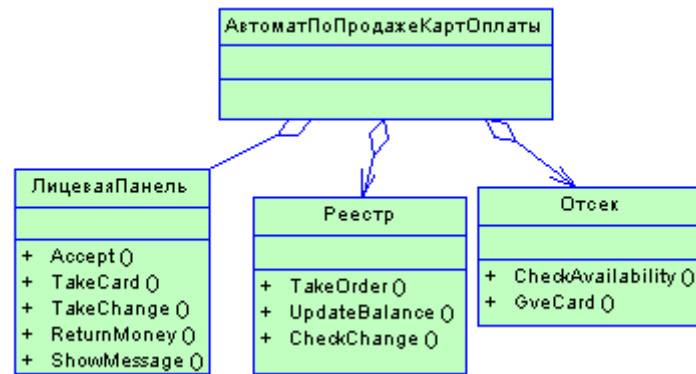


Рис. 14.2. Модифицированный вариант диаграммы классов

Рассмотрим первый сценарий.

1. Покупатель помещает деньги в щель на лицевой панели автомата и выбирает сорт лимонада.

2. Деньги попадают в реестр, который обрабатывает их.

3. Считаем, что в данном варианте выбранный сорт лимонада имеется в наличии и реестр дает команду отсеку доставить лимонад к лицевой панели.

Для выполнения рассматриваемого сценария выполним ряд операций:

1. Создадим диаграмму последовательности и назовем ее «Продажа карт оплаты».

2. С помощью инструмента *Object* создадим один объект. В свойствах в полях *Name* и *Code* уберем названия объекта, но в поле *Class* с помощью кнопки *Create* необходимо создать новый класс и в открывшемся окне со свойствами класс в поля *Name* и *Code* ввести слово «Клиент». Таким образом создается анонимный объект.

3. Далее в окне просмотра найдем папку *Classes*, раскроем ее, выделим класс «Лицевая панель» левой кнопкой мыши и, удерживая ее нажатой, перенесем этот класс из окна просмотра в рабочую область. Аналогично надо поступить с классами «Реестр» и «Отсек». Таким образом, нам не придется создавать заново объекты. Результат представлен на рис. 14.3.

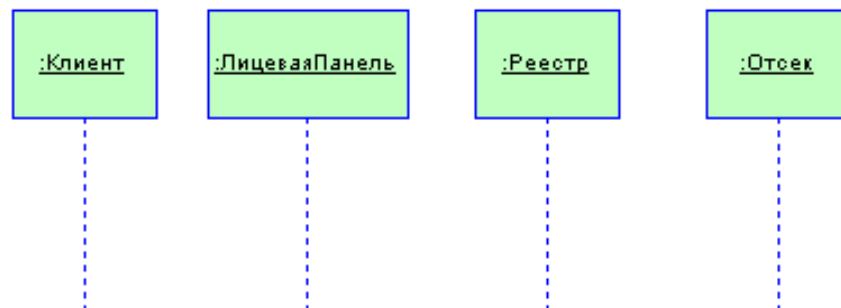


Рис. 14.3. Список классов на диаграмме последовательностей

4. Добавим асинхронные сообщения между объектами с помощью инструмента *Message*. Инструмент *Self Message* применяется для изображения асинхронного сообщения, которое объект посылает сам себе. Чтобы добавить

синхронное сообщение, применяются инструменты *Call Message* и *Self Call Message*. Для изображения ответного сообщения используются инструменты *Return Message* и *Self Return Message*. Для того чтобы показать длительность процесса активации, используется инструмент *Activation*. Теперь вернемся к примеру. Соединим созданные объекты с помощью инструментов и в соответствии с табл. 14.1. Результат представлен на рис. 14.4.

Таблица 14.1

Список сообщений между классами

Сообщение	Соединяемые объекты	Инструмент
Принять (сумма, выбор)	«Клиент» и «ЛицеваяПанель»	Call Message
Получить заказ Клиента (сумма, выбор)	«Лицевая панель» и «Реестр»	Call Message
Проверить наличие (выбор)	«Реестр» и «Отсек»	Call Message
Есть выбор	«Отсек» и «Реестр»	Return Message
Обновить денежный баланс (сумма, цена)	«Реестр» и «Реестр»	Call Message
Доставить карту оплаты лицевой панели	«Реестр» и «Отсек»	Call Message
Получить карту оплаты	«Отсек» и «ЛицеваяПанель»	Message

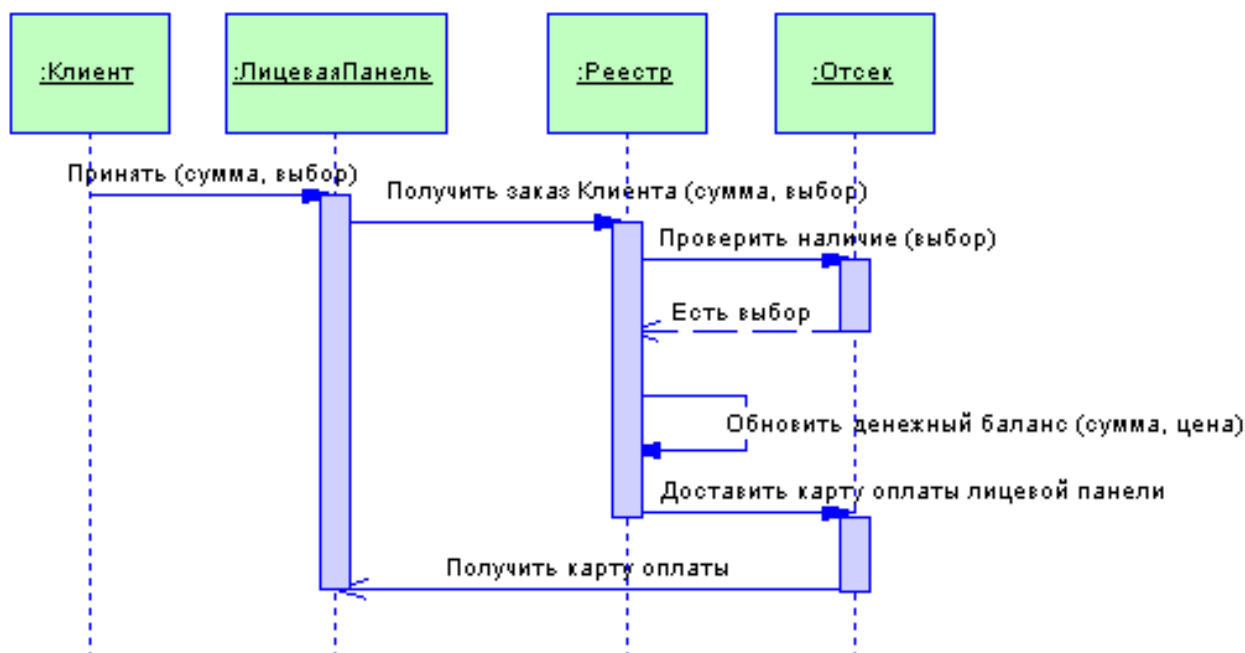


Рис. 14.4. Диаграмма последовательностей сценария № 1

Если диаграмма описывает только один сценарий прецедента, то это частная диаграмма последовательности (instance sequence diagram). По аналогии с первым сценарием надо разработать частные диаграммы последовательностей для остальных трех сценариев.

Второй сценарий: выбранный тип карты оплаты отсутствует (рис. 14.5).

Третий сценарий: покупатель вносит неточную сумму денег (рис. 14.6).

Четвертый сценарий: покупатель вносит неточную сумму денег при отсутствии сдачи в автомате (рис. 14.7).

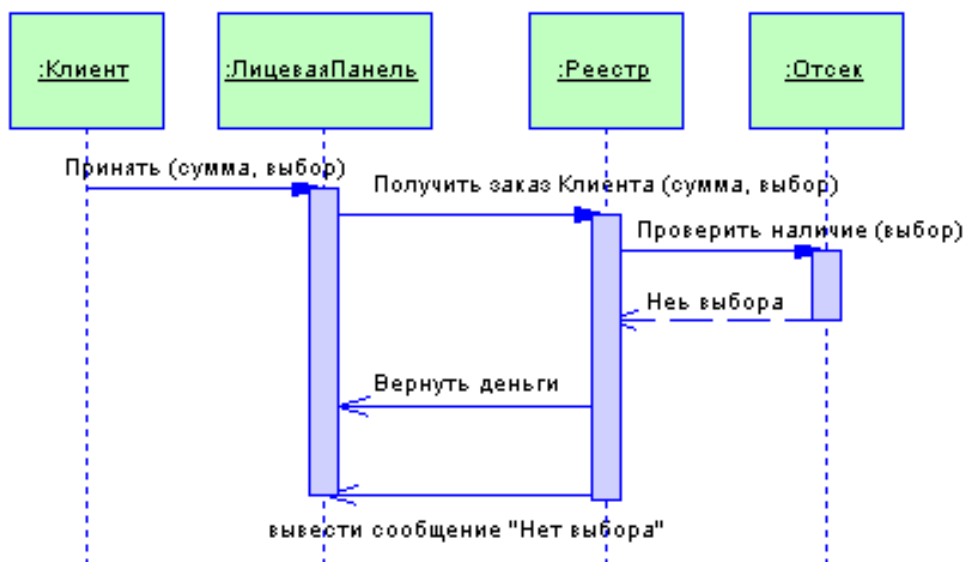


Рис. 14.5. Диаграмма последовательностей сценария № 2

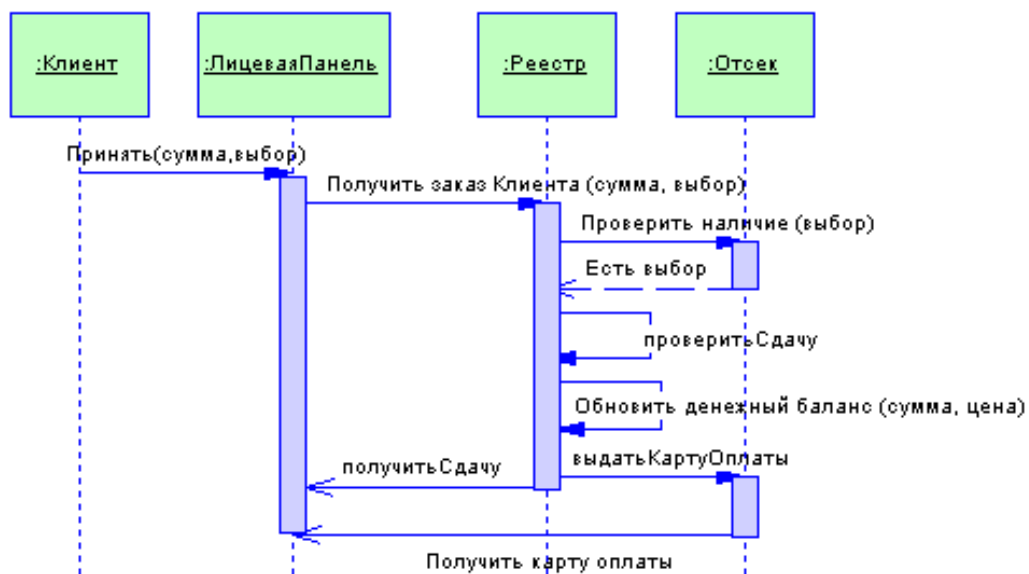


Рис. 14.6. Диаграмма последовательностей сценария № 3

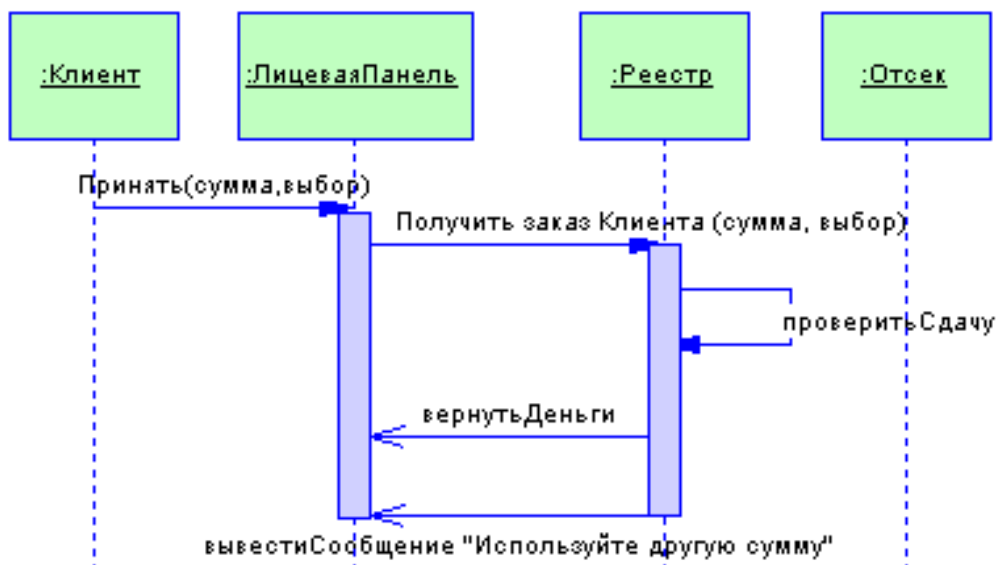


Рис. 14.7. Диаграмма последовательностей сценария № 4

Если диаграмма последовательности включает все сценарии прецедента, то она называется общей диаграммой последовательности. Учтем условия выполнения различных сценариев и построим общую диаграмму последовательностей (рис.14.8).

Пояснения к диаграмме: чтобы указать, что сообщение передается только в случае отсутствия нужного сорта лимонада, перед сообщением добавляют условие перехода «нет выбора». Для этого в свойствах во вкладке *Detail*, в поле *Condition* указывается нужное условие. Можно сделать это и с помощью ответного сообщения «нетВыбора». Но ответные сообщения на общей диаграмме последовательности будут только загромождать ее, поэтому их лучше не указывать.

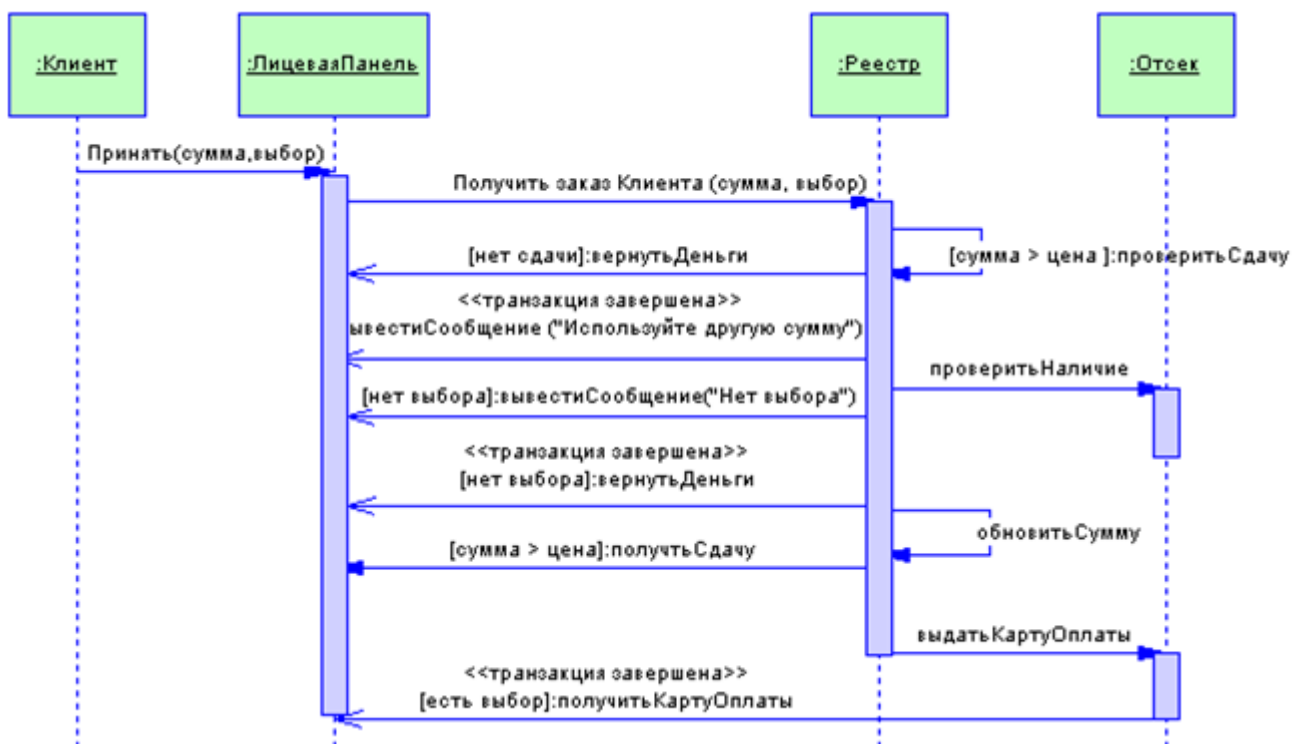


Рис. 14.8. Обобщенная диаграмма последовательностей с учетом всех сценариев

Контрольные вопросы

1. Назовите назначение диаграммы последовательностей.
2. Из каких объектов состоит диаграмма последовательностей?
3. Назовите типы сообщений, используемых в диаграмме последовательностей.
4. Для каких случаев в условии используется переход «нет выбора»?

Практическое занятие № 15

ТЕМА: «ДИАГРАММЫ КОММУНИКАЦИИ (COLLABORATION DIAGRAM)»

Цель занятия: изучить навыки создания диаграммы коммуникаций.

Задача: изучить навыки создания диаграммы коммуникаций.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка диаграммы коммуникаций информационной системы.

Теоретические основы занятия

На диаграммах коммуникации также отражается взаимодействие между объектами, но несколько иначе, чем на диаграммах последовательностей. На этих диаграммах изображаются объекты вместе с сообщениями, которыми они обмениваются.

Рассматриваемые диаграммы семантически эквивалентны. В них представлена одна и та же информация. Диаграмму последовательности можно преобразовать в диаграмму коммуникации и наоборот.

На самом деле удобно работать с обоими видами диаграмм. Диаграмма последовательностей акцентирует внимание на временном упорядочении взаимодействий. Диаграмма коммуникации ориентирована на состояние и общую организацию взаимодействующих объектов. Еще один признак отличия этих диаграмм – диаграмма последовательностей упорядочена в соответствии со временем, а диаграмма коммуникаций – в соответствии с пространственным расположением объектов.

Диаграмма коммуникации является расширенным понятием объектной диаграммы. Помимо связей между объектами здесь присутствуют также сообщения, которые объекты передают друг другу. Отличие диаграммы коммуникации от объектной диаграммы состоит в том, что в объектной диаграмме показаны экземпляры класса и их взаимосвязь в конкретный момент времени, а диаграмма коммуникации отражает взаимодействие объектов во времени.

Обращение к объекту представляется в виде стрелки, которая идет параллельно линии, соединяющей два объекта. Направление стрелки указывает на объект, к которому обращаются. Каждая стрелка имеет метку, в которой содержится описание передаваемого сообщения (передается команда от одного объекта другому на выполнение определенной операции). Чтобы было понятно, в какой последовательности передается информация, рядом с каждым сообщением стоит номер. Набор обозначений для диаграммы коммуникации представлен на рис. 15.1.

Для построения диаграммы коммуникаций используем диаграмму последовательности предыдущей практической работы № 14. Представим в Power-Designer первоначальный сценарий № 1 при продаже карт оплаты.

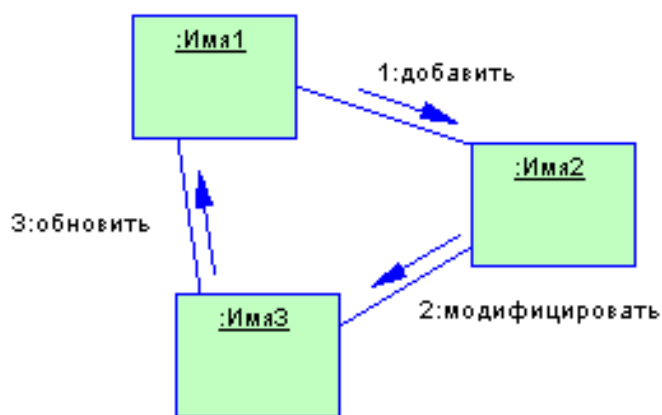


Рис. 15.1. Пример диаграммы коммуникаций

Экспериментальная часть занятия

1. Создать новую диаграмму: нажать в пустом поле правой кнопкой мыши *Diagram/New Diagram/Collaboration Diagram*. Дать название диаграмме – «Продажа карт экспресс-оплаты».

2. Далее в окне просмотра найти папку *Classes*, раскрыть ее, выделить класс «Клиент» левой кнопкой мыши и, удерживая ее нажатой, перенести этот класс из окна просмотра в рабочую область. Аналогично поступить с классами «Лицевая панель», «Реестр» и «Отсек». У каждого объекта открыть свойства, очистить поле *Name* и *Code*, чтобы объекты стали анонимными. Нажать кнопку *Применить* для сохранения изменений.

3. Создать сообщения между объектами в соответствии с табл. 15.1, используя инструмент *Message*. Чтобы открыть свойства сообщения, необходимо нажать на стрелку два раза левой кнопкой мыши. Чтобы указать какие данные передаются между объектами, необходимо открыть свойства сообщения и во вкладке *Detail* в поле *Arguments* ввести через запятую слова «Сумма» и «Выбор». Далее в поле *Operation* нажать кнопку *Create*. В открывшемся окне перейти на вкладку *Parameters* и ввести там параметры «Сумма» и «Выбор». Результат диаграммы представлен на рис. 15.2.

Таблица 15.1

Сообщения между классами

№ шага	Соединить объекты		Название сообщения
1	Клиент	Лицевая панель	принять(сумма, выбор)
2	Лицевая панель	Реестр	получитьЗаказКлиента(сумма, выбор)
3	Реестр	Отсек	проверитьНаличие(выбор)
3.1	Отсек	Реестр	Есть выбор
4	Реестр	Реестр	обновитьДенежныйБаланс(сумма, цена)
5	Реестр	Отсек	выдатьКартуОплаты(выбор)
6	Отсек	Лицевая панель	получитьКартуОплаты(выбор)

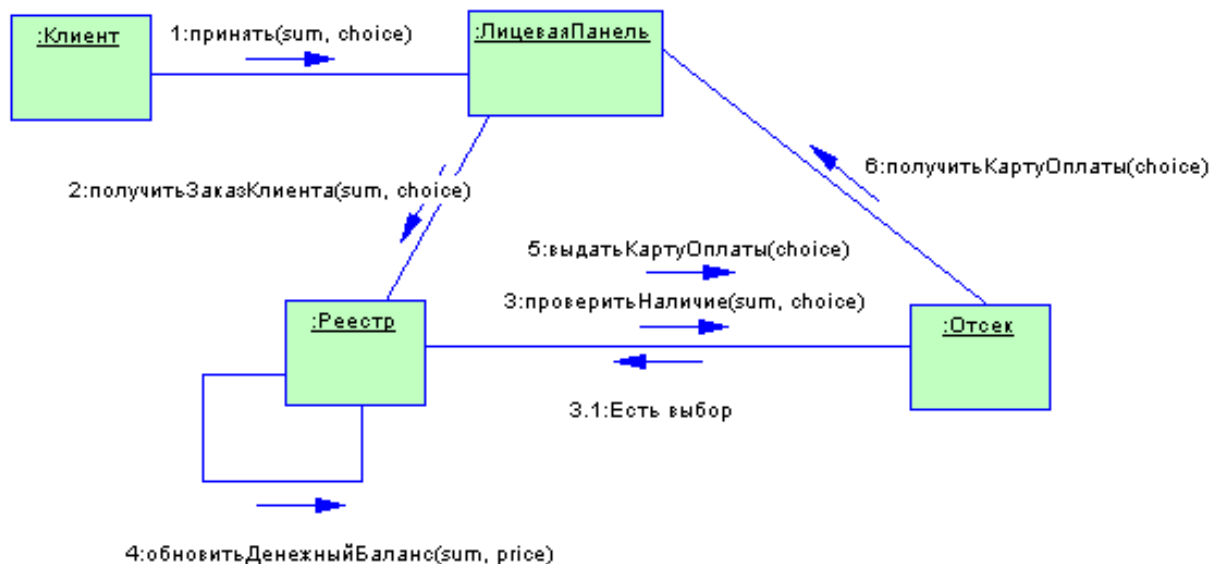


Рис. 15.2. Результат пункта 3 диаграммы коммуникаций

В диаграмме коммуникации существует еще один тип сообщения – вложенное сообщение, т.е. номер вложенного сообщения начинается с номера родительского сообщения, за которым следует собственно номер вложенного сообщения. В данном случае ответное сообщение «ЕстьВыбор» является вложенным для операции «проверить(выбор)», его номер – 3.1. Номера сообщений можно изменить в свойствах в поле *Sequence number*. Остальные частные диаграммы для всех перечисленных выше случаев строятся аналогично.

4. Для более ясного понимания данной диаграммы построим общую диаграмму с использованием условий перехода и стереотипов (рис. 15.3).

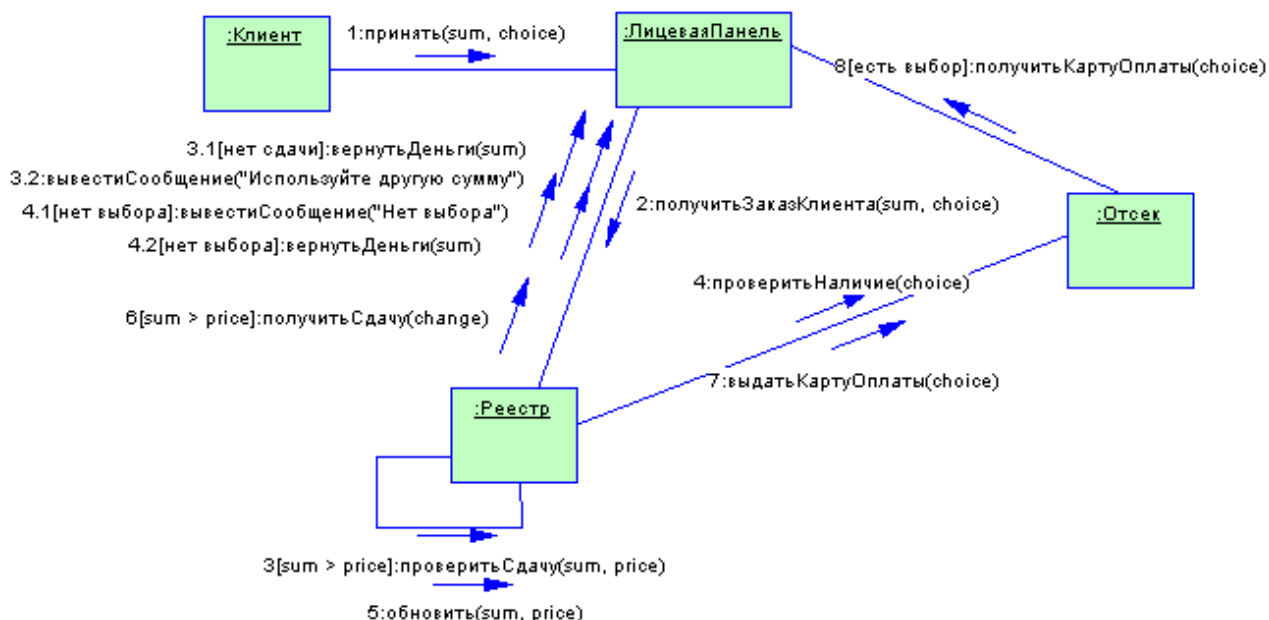


Рис. 15.3. Диаграмма коммуникаций с учетом условий перехода и стереотипов

5. В некоторых случаях объект передает сообщение только после отправки нескольких других непоследовательных сообщений. Чтобы изобразить данную ситуацию, воспользуемся примером, где сотрудники корпорации участвуют в продвижении нового продукта:

- вице-президент по маркетингу дает задание вице-президенту по продажам разработать рекламную кампанию для определенного вида продукции.
- вице-президент по продажам разрабатывает политику и поручает ее проведение менеджеру по продаже.
- менеджер по продаже направляет торгового агента вести торговлю продукцией согласно разработанной программе.
- агент устраивает распродажу в кругу потенциальных потребителей.
- после выполнения шагов 2 и 3 специалист корпорации по рекламе, согласно разработанной политике, размещает рекламные объявления в местной газете.

Создадим семь объектов с помощью инструмента *Object*. Каждый объект сделаем анонимным и в свойствах каждого объекта в поле *Classes* создадим новые классы в соответствии с табл. 15.2.

Таблица 15.2

Свойства сообщений

№ объекта	Поля Name и Code	Название класса
Объект_1	Пустые	ГлавныйВППоМаркетингу
Объект_2	Пустые	ВППоПродажам
Объект_3	Пустые	МенеджерПоПродажам
Объект_4	Пустые	ТорговыйАгент
Объект_5	Пустые	Покупатель
Объект_6	Пустые	СпециалистПоРекламе
Объект_7	Пустые	Газета

Сообщения между объектами должны быть размещены так, как показано на рис. 15.4.

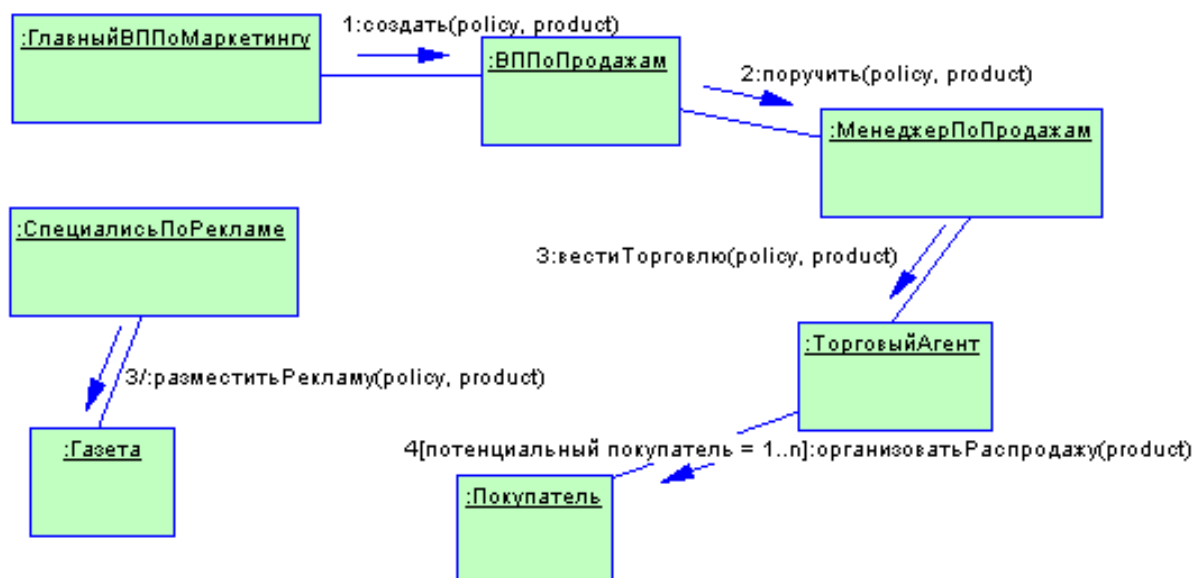


Рис. 15.4. Результирующая диаграмма коммуникаций

Контрольные вопросы

1. В чем отличие диаграммы коммуникаций от диаграммы последовательностей?
2. В каких случаях рекомендуется использовать диаграмму коммуникаций?

Практическое занятие № 16

ТЕМА: «ДИАГРАММЫ ВИДОВ ДЕЯТЕЛЬНОСТИ (ACTIVITY DIAGRAM)»

Цель занятия: изучить навыки создания диаграммы видов деятельности.

Задача: изучить навыки создания диаграммы видов деятельности.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка диаграммы видов деятельности информационной системы.


Теоретические основы занятия

Диаграмма видов деятельности является неотъемлемой частью системного анализа. Она похожа на блок-схемы, которые являются основой будущей программы (рис. 16.1). В данной диаграмме точками принятия решений и переходов описывается последовательность шагов. С помощью данной диаграммы, можно описать все, что происходит во время какой-либо операции или процесса. Каждый вид деятельности изображается в виде прямоугольника с округленными углами. После завершения одного вида деятельности переход к следующему происходит автоматически. Переход от одного вида деятельности к другому изображается стрелкой. На диаграмме видов деятельности имеются начальная точка (закрашенный кружок) и конечная точка (в виде «глазка»).



Рис. 16.1. Пример диаграммы видов деятельности

Точка принятия решения изображается двумя способами:

- показываются возможные пути развития ситуации после завершения данного вида деятельности;
- переход осуществляется с использованием ромба  (условие проверки в блок-схеме).

Если необходимо показать, что два вида деятельности происходят одновременно, а затем сходятся, то на диаграмме это показывается с помощью жирной непрерывной черной линии (рис.16.2). Слияние путей изображается еще одной линией (инструмент *Synchronization*).

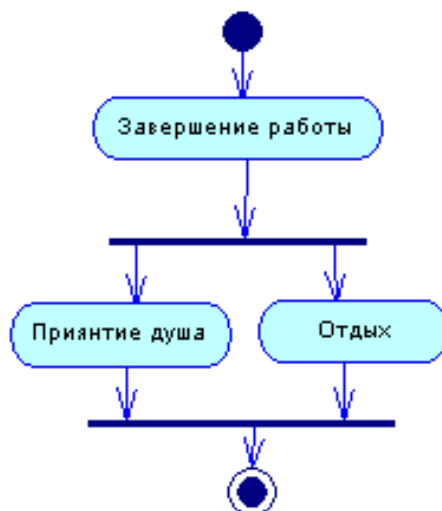


Рис. 16.2. Пример диаграммы видов деятельности с синхронизацией

Экспериментальная часть занятия

Необходимо создать диаграмму видов деятельности на примере сдачи функционала заказчику.

1. Создайте диаграмму видов деятельности: *Diagram/New Diagram/Activity Diagram*.

2. Создайте начальную точку с помощью инструмента *Start*.

3. Создайте семь видов деятельности с помощью инструмента *Activity*.

Мы договариваемся о встрече с клиентом, следовательно, Activity_1 назовем «Договор о встрече».

Если встреча состоится у нас, необходимо подготовить конференц-зал, если у заказчика – подготовить материалы к показу. Следовательно, Activity_2 – «Подготовка конференц-зала», Activity_3 – «Подготовка материалов».

Проводим показ функционала заказчику, следовательно, Activity_4 – «Встреча с заказчиком».

Далее следует оформление результатов встречи, следовательно, Activity_5 – «Оформление результатов встречи».

Если есть критичные замечания, то функционал не принимается, записываются замечания (Activity_6 – «Запись замечаний»), если функционал устраивает заказчика, то подписывается акт о приемке работ (Activity_7 – «Подпись акта о приемке работ»).

4. Создайте конечную точку с помощью инструмента *End*.

5. Изобразите переходы от одного вида деятельности к другому с помощью инструмента *Transition*. Если есть условия, их необходимо указать в свойствах перехода во вкладке *Condition*. На этапах принятия решения (условие «если») использовать инструмент *Decision*.

В итоге должна получиться диаграмма, представленная на рис. 16.3.

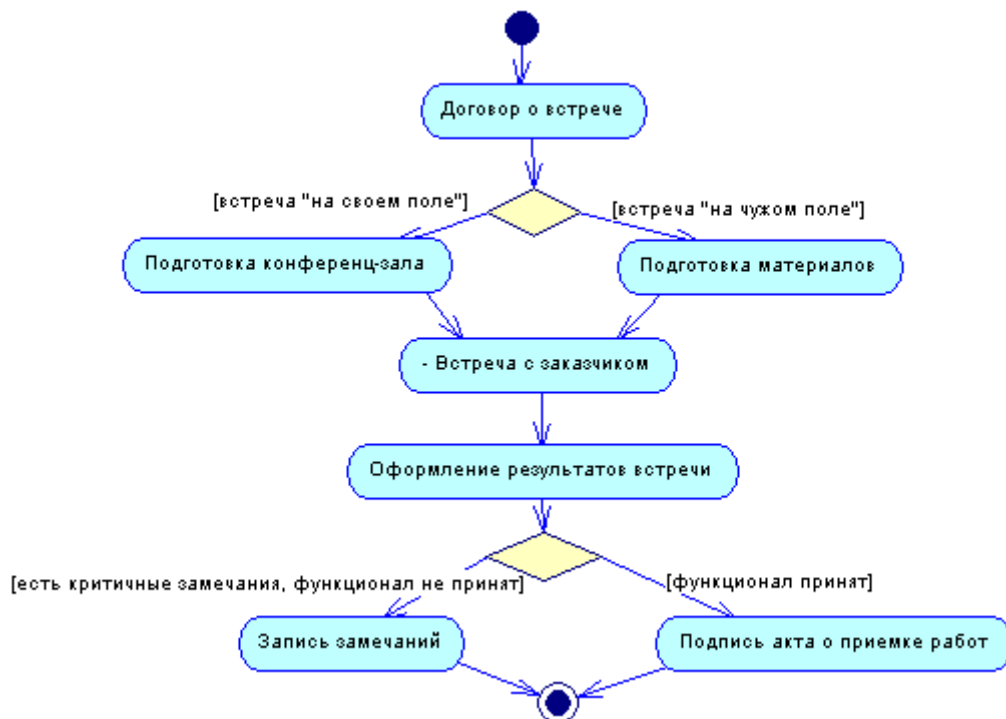


Рис. 16.3. Результирующая диаграмма видов деятельности


6. Важный аспект диаграммы видов деятельности – это возможность представить каждый объект диаграммы в виде новой диаграммы видов деятельности. Для этого необходимо нажать на нужный объект (например, объект «Запись замечаний») правой кнопкой мыши и выбрать *Change to Composite*. Справа внизу этого объекта появится значок, который говорит о том, что данную диаграмму возможно развернуть . Если нажать на данный объект правой кнопкой мыши и выбрать *Open Diagram*, то откроется чистый лист, на котором необходимо нарисовать шаги включенные в объект «Запись замечаний» (рис. 16.4).



Рис. 16.4. Последовательность шагов объекта «Запись замечаний»

7. Чтобы вернуть к главной диаграмме, необходимо нажать правую кнопку мыши в пустом месте рабочей области и выберите *Diagram/Go Up one Level* с указанием необходимой для открытия диаграммы.

8. Для отображения ролей участников процесса, диаграмма разбивается вертикальными линиями на сегменты. В верхней части каждого сегмента указывается название роли.

В PowerDesigner для обозначения сегментов используется инструмент *OrganizationUnit*. В каждом сегменте отображаются те виды деятельности, которые соответствуют данной роли. Разобьем по сегментам диаграмму, описывающую процесс сдачи функционала заказчику. Для этого необходимо:

- создать три сегмента. Названия сегментов «Начальник сектора», «Инженер (исполнитель)» и «Специалист по тех. обеспечению»;
- распределить виды деятельности по ролям (рис. 16.5).

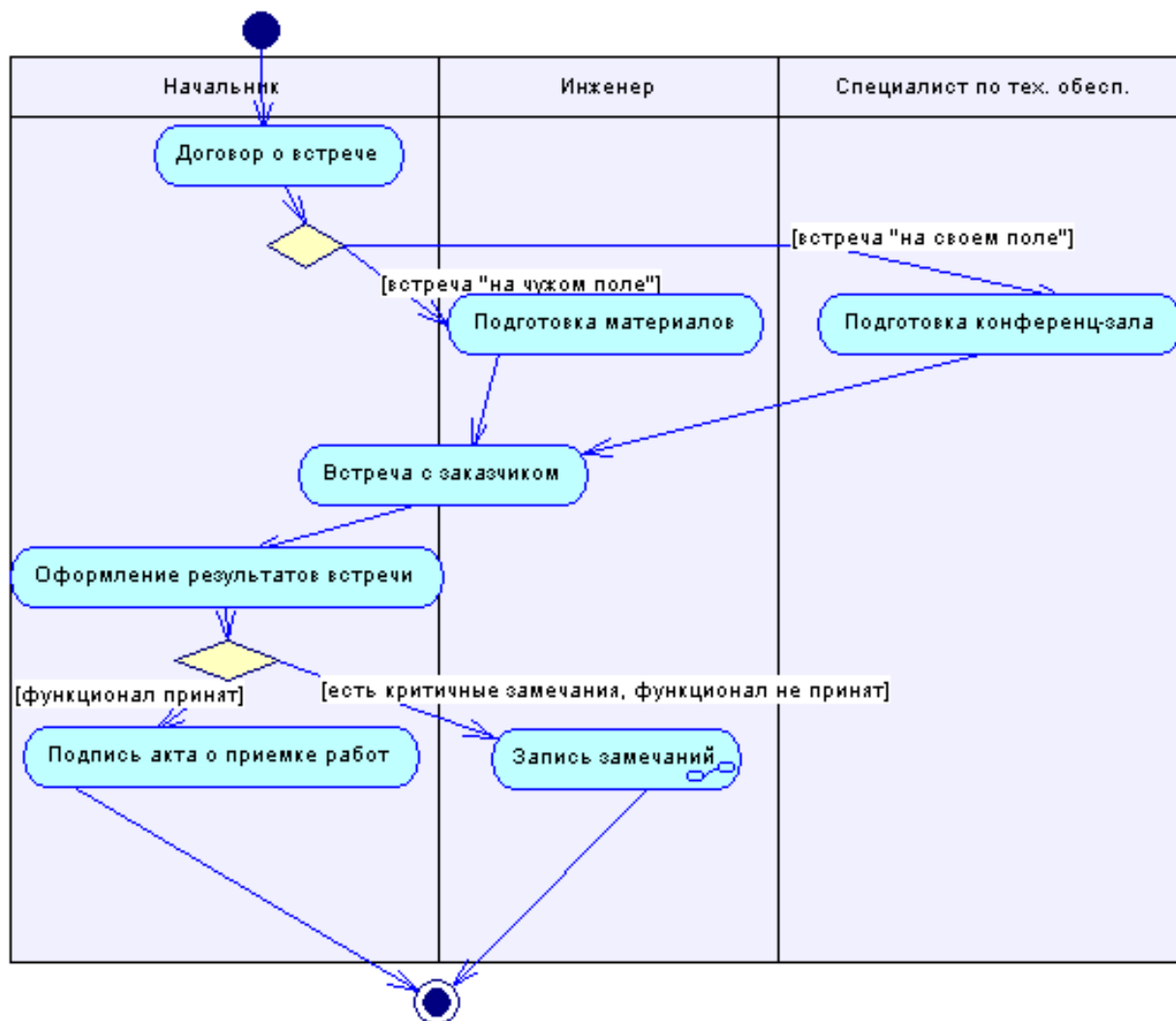


Рис. 16.5. Распределение видов деятельности по ролям

Контрольные вопросы

1. Назначение диаграммы видов деятельности.
2. Каким образом формируется композиция диаграммы?
3. Каким образом в диаграмме создаются роли?

Практическое занятие № 17

ТЕМА: «ДИАГРАММА КОМПОНЕНТОВ (COMPONENT DIAGRAM)»

Цель занятия: изучить навыки создания диаграммы компонентов.

Задача: изучить навыки создания диаграммы компонентов.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка диаграммы компонентов информационной системы.

Теоретические основы занятия

С помощью диаграммы компонентов описываются компоненты программного обеспечения. Программный компонент является частью системы. Он обеспечивает интерфейс с другими компонентами.

В UML 1.x компонентами считались таблицы, файлы данных, документы, исполняемые файлы и динамически подключаемые библиотеки. Для уточнения этих понятий, при моделировании эти компоненты назывались компонентами развертывания, рабочими и исполняемыми компонентами.

В UML 2.0 перечисленные понятия имеют одно имя – *артефакт* (фрагмент информации, используемый или генерируемый системой).

Компонент определяет функциональность системы, т.е. представляет реализацию одного или нескольких классов.

Артефакт – это реализация компонента.

Цели построения модели компонентов следующие:

- заказчик сможет увидеть структуру законченной системы;
- разработчики смогут представить себе структуру будущей системы;
- редакторы, ответственные за написание инструкций и справочной документации, смогут лучше понять суть разработки.

Компоненты можно использовать в дальнейшем многократно.

При работе с компонентами используется интерфейс. Операции компонента выполняются только через интерфейс. Связь между компонентом и интерфейсом называется реализацией. Интерфейс компонента может быть открытым, и операции этого интерфейса могут использоваться другими компонентами. Другими словами, компонент может получать доступ к услугам другого компонента. Компонент, обеспечивающий доступ, называется экспортируемый интерфейс. Компонент, который пользуется этим доступом, называется импортируемый интерфейс.

Один компонент можно заменить другим, если новый компонент имеет такой же интерфейс. Разработчику, который пытается заменить или повторно использовать компонент, будет намного удобнее, если информация компонентного интерфейса легко доступна в форме модели, если нет, то разработчик вынужден идти более длинным путем обратного проектирования (реинжиниринга) на основе кода.

Экспериментальная часть занятия

1. Создайте диаграмму компонентов: *Diagram/New Diagram/Component Diagram*.

2. Диаграмма компонентов содержит компоненты, интерфейсы и их взаимосвязи. Данная диаграмма обозначается в виде прямоугольника, на левую сторону которого наложены еще два прямоугольника (рис. 17.1).

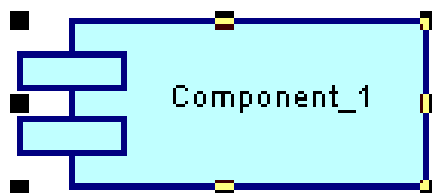




Рис. 17.1. Пример диаграммы компонентов

3. В свойствах компонента указываются имя (поля *Name* и *Code*), стереотип (поле *Stereotype*), в поле *Type* указывается тип компонента (стандартный или специальный компонент, включающий EJB, JSP, Servlet).

Чтобы создать новый интерфейс, необходимо перейти на вкладку *Interfaces* в свойствах компонента. Вы можете создать новый интерфейс (кнопка ) или добавить уже ранее созданные интерфейсы (кнопка )

При создании нового интерфейса откроется окно со свойствами интерфейса, где необходимо ввести имя, добавить атрибуты и операции.

Создайте компонент *Calculator* и добавьте ему интерфейс (рис. 17.2).

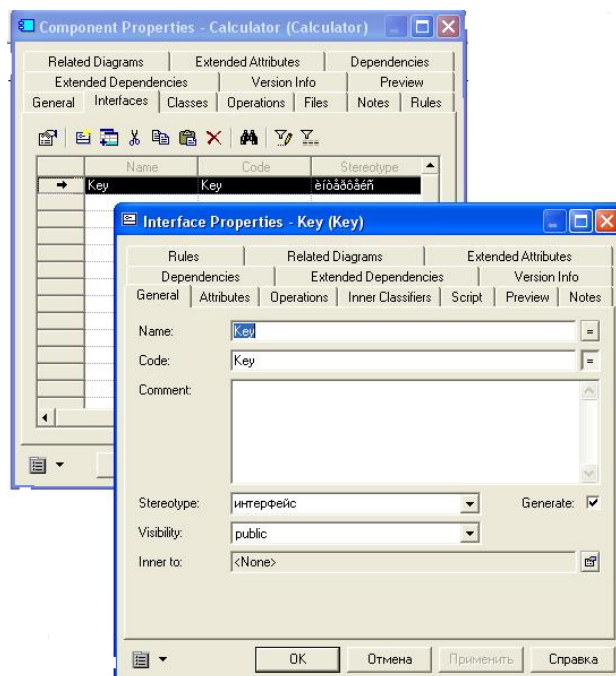


Рис. 17.2. Пример создания компонента и интерфейса

В результате получим диаграмму, представленную на рис. 17.3.

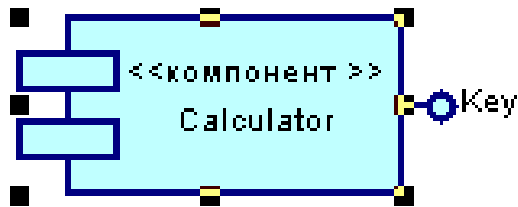




Рис. 17.3. Пример диаграммы с интерфейсом

Компонент может содержать классы (это классы, обеспечивающие выполнение компонента). Обычно один класс является главным, а другие используются для выполнения функций компонента. Классы не имеют обозначения в диаграмме компонентов (как, например, интерфейсы). Но отношение между классом и компонентом можно увидеть в свойствах компонента на вкладке *Classes*. Причем вы можете добавить либо новый класс (кнопка ) , либо выбрать ранее созданный в другой диаграмме класс (кнопка ).

Вы можете добавить файл (вкладка *Files* в свойствах компонента). Во вкладке *Operations* можно увидеть операции, которые были объявлены в свойствах интерфейса компонента.

Вы можете создать диаграмму классов для выбранного компонента, чтобы более детально рассмотреть классы и интерфейсы. Для этого, нажмите на компонент правой кнопкой мыши и выберите из меню *Create/Update Class Diagram*.

Вы можете создать компонент из диаграммы классов. Для этого перейдите в рабочее пространство диаграммы классов и в меню выберите *Tools/Create Component*, далее следуйте указаниям помощника.

Чтобы показать взаимосвязь между артефактом (например, исполняемым кодом) и реализующим его компонентом, используется инструмент *Generalization* и в свойствах указывается стереотип *<<Реализует>>* (рис. 17.4).

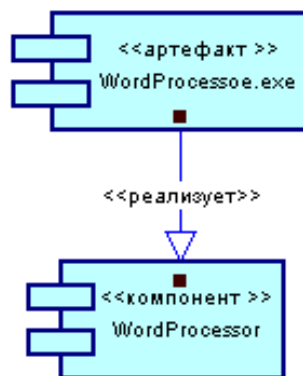


Рис. 17.4. Пример связи реализации между артефактом и компонентом

Зависимость между компонентами изображается с помощью инструмента *Dependency*.

Контрольные вопросы

1. Назначение диаграммы компонентов.
2. Из каких элементов состоит диаграмма компонентов?
3. Приведите примеры типов компонентов.

Практическое занятие № 18

ТЕМА: «ДИАГРАММА РАЗВЕРТЫВАНИЯ (DEPLOYMENT DIAGRAM) И ДИАГРАММА ПАКЕТОВ (PACKAGES DIAGRAM)»

Цель занятия: изучить навыки создания диаграммы развертывания и диаграммы пакетов.

Задачи:

1. Изучить навыки создания диаграммы развертывания.
2. Изучить навыки создания диаграммы пакетов.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка диаграммы развертывания информационной системы.
3. Разработка диаграммы пакетов информационной системы.

Теоретические основы занятия

Диаграмма развертывания (Deployment Diagram)

Аппаратные средства очень важны в многокомплексных системах. В современном компьютерном мире такие системы являются распределенными, предоставляют большие возможности и могут использоваться на множестве различных платформ. Вопросы развертывания аппаратных средств должны быть хорошо проработаны еще в процессе проектирования. Язык UML предоставляет систему обозначений для создания проекта развертывания аппаратных средств.

Диаграммы развертывания отображают способ воплощения артефактов в физической системе и способ соединения аппаратных средств между собой. Главным аппаратным элементом является узел – общее название для любого вычислительного ресурса.

В UML 1.x выделяли два типа узлов:

- процессоры – узлы, выполняющие команды компонента;
- устройства – периферийные аппаратные средства, которые не выполняют команды компонентов, а осуществляют интерфейс с внешним миром.

В UML 2.0 устройство формально определяется как узел, выполняющий артефакты (исполняемый код).

Диаграмма пакетов (Packages Diagram)

Данная диаграмма помогает более глубоко понять другие диаграммы.

Пакет предназначен для группирования элементов диаграмм (рис. 18.1). В терминах языка UML пакет предоставляет для содержащихся в нем элементов пространство имен. Пакеты могут связываться друг с другом одним из трех способов:

- один пакет может обобщать другой пакет;
- один пакет может зависеть от другого пакета;

- один пакет может уточнять другой пакет.

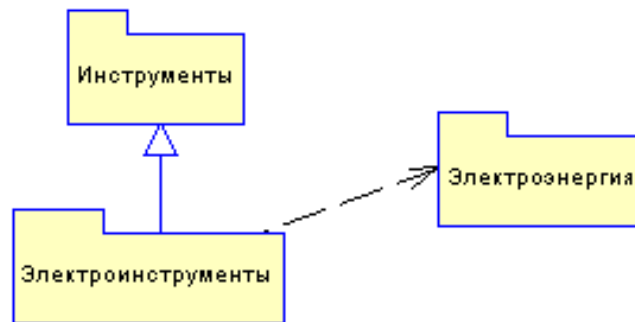


Рис. 18.1. Отношения обобщения и зависимости между пакетам

Экспериментальная часть занятия

Диаграмма развертывания

Узел изображается в виде куба, с которым связано определенное имя и необязательное ключевое слово <<устройство>> (рис. 18.2).

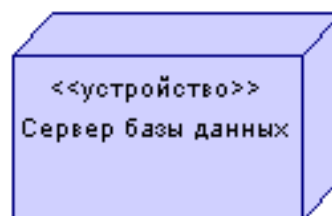


Рис. 18.2. Пример узла «Сервер базы данных»

В PowerDesigner для изображения узла используется инструмент *Node*. Для соединения узлов, используется инструмент *Node Association*.

В UML 2.0 появилось новое понятие – спецификация развертывания – артефакт, обеспечивающий параметры для другого артефакта. Для изображения спецификации развертывания используется инструмент *Dependency*.

Рассмотрим несколько примеров.

1. *Кольцевая сеть с маркерным доступом*. В кольцевой сети с маркерным доступом компьютеры, оснащенные сетевыми адаптерами, подсоединены к центральному устройству множественного доступа. Многочисленные устройства соединены в кольцо. Кольцо действует подобно регулировщику, использующему сигнал, называемый маркером, чтобы дать каждому компьютеру знак, когда он может передавать информацию.

Когда компьютер получает маркер, только его информация может направляться в сеть. После отправки информация передается по значению. Когда она достигает пункта назначения, компьютеру, с которого ее отправили, передается уведомление о получении информации. В PowerDesigner данный пример сети представлен на рис. 18.3.

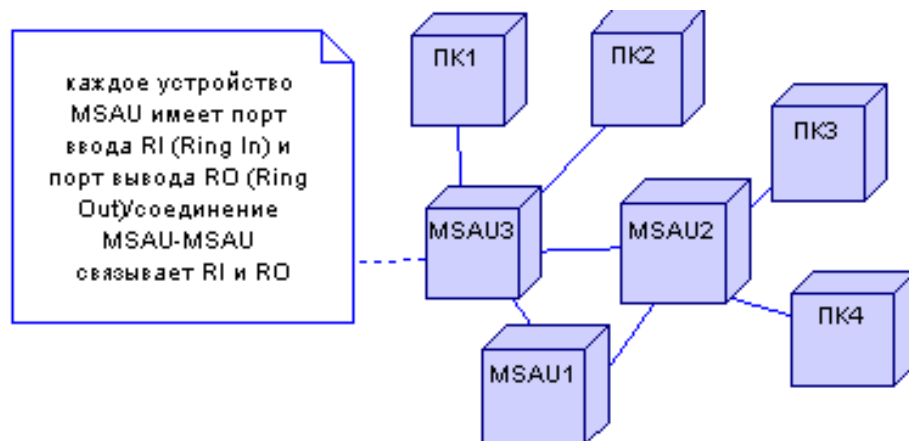


Рис. 18.3. Пример «Кольцевая сеть»

2. *Сеть ARC*. Это – сеть с подключаемыми ресурсами, которая осуществляет передачу маркера от компьютера к компьютеру. Но в этой сети каждый компьютер имеет собственный номер, с помощью которого определяется, какой из компьютеров получает маркер. Каждый компьютер соединен с концентратором, который может быть активным (восстанавливает и ретранслирует сигнал) и пассивным (просто выполняет коммутацию). В отличие от устройств MSAU в маркерном кольце концентраторы ARC не перемещают маркер по кольцу, а компьютеры пересылают маркер друг другу (рис. 18.4).

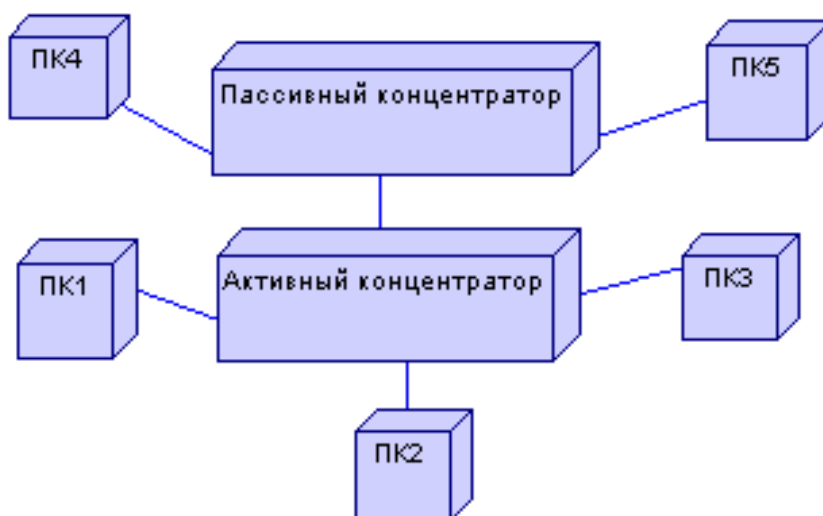


Рис. 18.4. Пример «Сеть ARC»

Диаграмма пакетов

В PowerDesigner пакеты можно создать в любой диаграмме. Для этого применяется инструмент *Package*. При этом в *Workspace* автоматически создается пакет, в который вложена та диаграмма, где был создан пакет. Пакет также можно создать, если в *Workspace* нажать на любой созданной ранее диаграмме правой кнопкой мыши и далее выбрать *Convert to package*. В результате будет создан пакет, а выбранная диаграмма переместится автоматически в папку *Package*.

Для отображения обобщения используется инструмент *Generalization*, зависимости – инструмент *Dependency*, уточнения – инструмент *Dependency* и в свойствах в поле *Stereotype* прописывается «уточняет».

Пакет может объединяться с другим пакетом. Отношение объединения – это отношение зависимости, которое объединяет пакет-источник с целевым пакетом. В результате объединения получается преобразование пакета-источника.

Контрольные вопросы

1. Назовите назначение диаграммы пакетов.
2. Назовите назначение диаграммы развертывания.
3. В каких случаях необходима диаграмма развертывания?
4. Какие существуют способы связывания между пакетами?

Практическое занятие № 19

ТЕМА: «МЕТОДОЛОГИЯ ТЕСТИРОВАНИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ»

Цель занятия: приобретение основных навыков в разработке системы модульного тестирования.

Задача: изучить методологию модульного тестирования информационной системы на примере среды программирования Qt Creator.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Разработка системы модульного тестирования в среде программирования Qt Creator.

Теоретические основы занятия

Инструментарий QTestLib является средством для тестирования приложений и библиотек Qt. QTestLib предоставляет полный базовый функционал для тестирования графического пользовательского интерфейса.

Возможности QTestLib. QTestLib создан для облегчения написания тестов для приложений и библиотек Qt (табл. 19.1).

Таблица 19.1

Возможности QTestLib

Возможности	Детали
Легковесность	QTestLib содержит порядка 6000 строк кода и 60 экспортируемых символов
Самодостаточность	QTestLib нуждается только в нескольких символах из Qt Core библиотеки для тестирования без GUI
Быстрое тестирование	QTestLib не требует запуска специальных тестов
Ориентированное на данные тестирование	Тестирование может быть выполнено в любое время с разными данными
Базовое тестирование графического интерфейса пользователя	QTestLib предлагает функционал симуляции сообщений от мыши и клавиатуры
Проведение эталонных тестов (Benchmarking)	QTestLib поддерживает проведение эталонных тестов и предоставляет несколько измерительных бэкендов
Дружественность к IDE	QTestLib выводит сообщения, которые могут быть обработаны в Visual Studio и KDevelop
Потокобезопасность	Отчет об ошибках атомарен и потокобезопасен
Безопасность с точки зрения типов	Исчерпывающее использование шаблонов предотвращения ошибок, внесенных неявной проверкой типов
Легкая расширяемость	Пользовательские типы могут быть легко добавлены в тестируемые данные и тестовый вывод данных

Создание теста. Для создания теста переопределите QObject и добавьте один или несколько закрытых слотов в нем. Каждый закрытый слот является функцией вашего теста. QTest::qExec() используется для выполнения всех функций тестов в объекте-тесте.

Дополнительно нужно добавить четыре закрытых слота, которые не будут выполнять роль функций тестов. Они запускаются структурой тестирования и могут быть использованы для инициализации и очистки любого из тестов или текущей функции-теста.

- `initTestCase()` выполняется перед запуском первой функции-теста;
- `cleanupTestCase()` выполняется после выполнения последней функции-теста;
- `init()` выполняется перед запуском каждой функции-теста;
- `cleanup()` выполняется после каждой функции-теста.

Если `initTestCase()` вернул ошибку, значит, нет функций тестов, которые могут быть запущены. Если `init()` вернул ошибку, значит, функция-тест не была запущена, после чего тест перейдет к следующей функции-тесту. Пример:

```
class MyFirstTest: public QObject
{
    Q_OBJECT
private slots:
    void initTestCase()
    { qDebug("called before everything else"); }
    void myFirstTest()
    { QVERIFY(1 == 1); }
    void mySecondTest()
    { QVERIFY(1 != 2); }
    void cleanupTestCase()
    { qDebug("called after myFirstTest and mySecondTest"); }
};
```

Сборка теста. Если вы используете `qmake` в качестве инструмента сборки, то добавьте следующую строку в ваш файл проекта: **QT += testlib**.

Если вы используете другой инструмент сборки, то убедитесь, что вы добавили путь к заголовочным файлам `QTestLib` (обычно `include/QtTest` в папке установки Qt). Если вы используете релиз-сборку Qt, то свяжите ваш тест с библиотекой `QtTest`. Для отладочных сборок используйте `QtTest_debug`.

Аргументы командной строки. Синтаксис выполнения автотеста имеет такую форму:

```
testname [options] [testfunctions[:testdata]]...
```

Замените `testname` реальным именем исполняемого файла, `testfunctions` может содержать имена функций-тестов, которые должны выполняться. Если `testfunctions` не задана, то будут выполнены все тесты. Если вы добавите имя записи в `testdata`, то функция-тест будет выполнена только с этими тестовыми данными. Полный перечень аргументов представлен в табл. 19.2. Рассмотрим несколько примеров.

Производится запуск функции-теста метода `toUpper` со всеми тестовыми данными:


```
/myTestDirectory$ testQString toUpper
```

Производится запуск функции-теста toUpper со всеми тестовыми данными и функции toInt с тестовыми данными с именем zero:

```
/myTestDirectory$ testQString toUpper toInt:zero
```

Производится запуск функции-теста testMyWidget, которая выводит информацию о возникновении любого сигнала и ждет 500 мс после каждой симуляции событий мыши/клавиатуры:

```
/myTestDirectory$ testMyWidget -vs -eventdelay 500
```

Таблица 19.2

Аргументы командной строки

Аргумент	Описание
-help	Вывод доступных аргументов командной строки и получение помощи
-functions	Вывод всех доступных функций теста
-o <i>filename</i>	Запись выводимых данных в файл
-silent	Вывод показывает только предупреждения, ошибки и минимальное статус-сообщения
-v1	Выводит информацию от введенных и существующих функций тестов
-v2	Расширяет вывод, также выводит QCOMPARE() и QVERIFY()
-vs	Выводит информацию о возникновении любого сигнала
-xml	Вывод в XML-формате вместо обычного текста
-lightxml	Выводит результаты как XML поток
-eventdelay <i>ms</i>	Если не определена задержка для симуляции клавиатуры или мыши (QTest::keyClick(), QTest::mouseClick() и т.д.), будет использовано значение этого параметра (в мс)
-keydelay <i>ms</i>	Похоже на -eventdelay, но включает только симуляцию клавиатуры, без мыши
-mousedelay <i>ms</i>	Похоже на -eventdelay, но включает только симуляцию мыши, без клавиатуры
-keyevent-verbose	Более подробный вывод при симуляции клавиатуры
-maxwarnings <i>numberBR</i>	Устанавливает максимальное число предупреждений, записываемых в вывод. 0 - не ограничивать, по умолчанию установлено в 2000

Создание эталонного теста. Для создания эталонного теста следуйте инструкциям по созданию теста и затем добавьте макрос QBENCHMARK в функцию теста, с которой вы хотите проводить эталонный тест. Пример:

```
class MyFirstBenchmark: public QObject
{ Q_OBJECT
private slots:
    void myFirstBenchmark()
    {
        QString string1;
        QString string2;
        QBENCHMARK {
            string1.localeAwareCompare(string2);
        }
    }
};
```

Код внутри макроса QBENCHMARK будет измерен и, возможно, также повторен несколько раз для того, чтобы получить точное измерение. Это зависит от выбранного измерительного инструмента. Доступны и могут быть выбраны из командной строки несколько инструментов (табл. 19.3).

Таблица 19.3

Измерительные инструменты эталонного теста

Имя	Аргумент командной строки	Доступность
Walltime	(по умолчанию)	Все платформы
Счетчик тактов ЦПУ	-tickcounter	Windows, Mac OS X, Linux, многие UNIX-системы.
Valgrind/Callgrind	-callgrind	Linux (требуется установка)
Счетчик событий	-eventcounter	Все платформы

Walltime доступен на всех платформах, но требует достаточно большого числа повторов для получения адекватного результата.

Счетчики тактов обычно доступны и могут предоставить результаты после нескольких повторов, но могут быть чувствительны к проблемам масштабирования частоты ЦПУ.

Valgrind предоставляет точные результаты, но не учитывает задержки ввода/вывода, и доступен только на ограниченном количестве платформ.

Подсчет событий доступен на всех платформах и предоставляет количество событий, которые были приняты от цикла обработки событий до их отправки соответствующим получателям (это может включать в себя не только события Qt).

Экспериментальная часть занятия

Задание: реализовать класс Smart, который обеспечивает операции сравнения целых чисел. В состав класса должны входить методы для определения минимального **int min(int, int)** и максимального **int max(int, int)** из двух чисел.

1. Открываем приложение Qt Creator. Создаем консольное приложение Qt. Добавляем к проекту модуль testlib. Для тестирования графического интерфейса следует добавить в файл проекта модуль gui:

max_min.pro

```
QT      += core testlib
QT      -= gui
TARGET = max_min
CONFIG  += console
CONFIG  -= app_bundle
TEMPLATE = app
SOURCES += main.cpp \
            smart.cpp \
            test_smart.cpp
HEADERS += \
            smart.h \
            test_smart.h
```

2. Добавляем в проект класс Smart, реализуем методы:

smart.h

```
#ifndef SMART_H
#define SMART_H
#include <QObject>
class Smart : public QObject
{
    Q_OBJECT
public:
    explicit Smart(QObject *parent = 0);
public slots:
    int max(int a, int b);
    int min(int a, int b);
};
#endif // SMART_H
```

smart.cpp

```
#include "smart.h"
Smart::Smart(QObject *parent) :
    QObject(parent)
{}
int Smart::max(int a, int b)
{
    if(a > b)
        return a;
    return b;
}
int Smart::min(int a, int b)
{
    if(a < b)
        return a;
    return b;
}
```

3. Создаем класс **TestSmart** для тестирования класса программы:

test_smart.h

```
#ifndef TEST_SMART_H
#define TEST_SMART_H
#include <QObject>
class TestSmart : public QObject
{
    Q_OBJECT
public:
    explicit TestSmart(QObject *parent = 0);
private slots:    //== должны быть приватными
    void max();    //== тест для проверки функции int max(int a, int b)
    void min();
    void min_data();
};
#endif // TEST_SMART_H
```

test_smart.cpp

```
#include <QTest>
#include "test_smart.h"
#include "smart.h"
TestSmart::TestSmart(QObject *parent) :
    QObject(parent)
{}
void TestSmart::max()
{
    Smart a;
    QCOMPARE(a.max(1, 0), 1);
    QCOMPARE(a.max(-1, 1), 1);
    QCOMPARE(a.max(4, 8), 8);
    QCOMPARE(a.max(0, 0), 0);
    QCOMPARE(a.max(1, 1), 1);
    QCOMPARE(a.max(-10, -5), -5);
}
void TestSmart::min_data()
{}
void TestSmart::min()
{}

```

4. В файле **main.cpp** добавим программный код для запуска системы тестирования:

main.cpp

```
#include <QtCore/QCoreApplication>
#include <QTest>
#include <iostream>
#include <cstdlib>
#include <cstdio>
#include "test_smart.h"
using namespace std;
int main(int argc, char *argv[])
{
    freopen("testing.log", "w", stdout);
    QCoreApplication a(argc, argv);
    QTest::qExec(new TestSmart, argc, argv);
    return 0;
}

```

5. Компилируем и запускаем программу. Изучаем содержимое лог-файла с результатами прохождения теста (результат показать преподавателю):

testing.log

```
***** Start testing of TestSmart *****
Config: Using QTest library 5.5.1, Qt 5.5.1
PASS    : TestSmart::initTestCase()
PASS    : TestSmart::max()
PASS    : TestSmart::min()
PASS    : TestSmart::cleanupTestCase()
Totals: 4 passed, 0 failed, 0 skipped, 0 blacklisted
***** Finished testing of TestSmart *****

```

6. Изменяем далее программный код, добавляем новый метод «табличка» для тестирования **min_data()** в файл в **test_smart.cpp**:

```
void TestSmart::min_data()
{
    QTest::addColumn<int>("first");
    QTest::addColumn<int>("second");
    QTest::addColumn<int>("result");
    QTest::newRow("min_data_1") << 1 << 0 << 0;
    QTest::newRow("min_data_2") << -1 << 1 << -1;
    QTest::newRow("min_data_3") << 4 << 8 << 4;
    QTest::newRow("min_data_4") << 0 << 0 << 0;
    QTest::newRow("min_data_5") << 1 << 1 << 1;
    QTest::newRow("min_data_6") << -10 << -5 << -10;
}

void TestSmart::min()
{
    Smart a;
    QFETCH(int, first);
    QFETCH(int, second);
    QFETCH(int, result);
    QCOMPARE(a.min(first, second), result);
}
```

7. Компилируем и запускаем программу. Анализируем лог-файл **testing.log**.

8. Проверяем систему тестирования на поиск ошибок в коде программы.

Например, в методе **Smart::min()** поменяем строку «**if(a < b)**» на «**if(a > b)**».

9. Компилируем и запускаем программу. Проанализируем лог-файл.

10. Выполняем тестирование графического интерфейса на примере визуального компонента **QLineEdit**. Добавляем в проект класс **test_qlineedit**:

test_qlineedit.h

```
#ifndef TEST_QLINEEDIT_H
#define TEST_QLINEEDIT_H
#include <QObject>
class Test_QLineEdit : public QObject
{
    Q_OBJECT
private slots: // должны быть приватными
    void edit();
};
#endif // TEST_QLINEEDIT_H
```

test_qlineedit.cpp

```
#include <QtTest>
#include <QtGui>
#include "test_qlineedit.h"
void Test_QLineEdit::edit()
{
    QLineEdit a;
    QTest::keyClicks(&a, "abCDEf123-");
    QCOMPARE(a.text(), QString("abCDEf123-"));
    QVERIFY(a.isModified());
}
```

11. Изменяем содержимое метода **main()** в файле **main.cpp**:

```
#include <QApplication>
#include <QTest>
#include <iostream>
#include <cstdlib>
#include <stdio>
#include "test_smart.h"
#include "test_qlineedit.h"
using namespace std;
int main(int argc, char *argv[])
{
    freopen("testing.log", "w", stdout);
    QApplication a(argc, argv);
    QTest::qExec(new TestSmart, argc, argv);
    cout << endl;
    QTest::qExec(new Test_QLineEdit, argc, argv);
    return 0;
}
```

12. Компилируем и запускаем программу. Оцениваем правильность работы визуального компонента по лог-файлу:

```
***** Start testing of Test_Smart *****
Config: Using QtTest library 5.5.1, Qt 5.5.1
PASS   : Test_QLineEdit::initTestCase()
PASS   : Test_QLineEdit::edit()
PASS   : Test_QLineEdit::cleanupTestCase()
Totals: 3 passed, 0 failed, 0 skipped
***** Finished testing of Test_QLineEdit *****
```

Контрольные вопросы

1. Перечислите основные методики тестирования информационных систем.
2. Перечислите основные возможности использования библиотеки тестирования QTestLib.
3. Как влияет библиотека тестирования QTestLib на работоспособность тестируемого приложения?

Практическое занятие № 20

ТЕМА: «СОВМЕСТНАЯ РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ»

Цель занятия: приобретение основных навыков в использовании системы контроля версий.

Задача: изучить систему контроля версий для совместной работы.

План занятия:

1. Ознакомление с содержанием теоретической части.
2. Отработать основные команды системы контроля версий.

Теоретические основы занятия

Система контроля (управления) версиями (от англ. Version Control System (VCS) или Revision Control System) – это программное обеспечение для облегчения работы с изменяющейся информацией.

Основные функции системы контроля версий следующие:

- хранение нескольких версий одного и того же документа (история версий);
- хранение истории разработки;
- при необходимости возвращение к более ранним версиям документа (отмена изменений);
- определение, кто и когда сделал изменение (поиск «виновного»);
- совмещение изменений сделанных разными разработчиками (синхронизация работы команды);
- реализация альтернативных/экспериментальных вариантов проекта.

По степени централизации системы контроля версий условно делятся на следующие типы:

- локальные (например, RCS);
- централизованные (например, CVS, Subversion и Perforce);
- децентрализованные (например, Git, Mercurial, Bazaar, Darcs).

Централизованные системы контроля версий. Особенностью таких систем является то, что они позволяют организовать процесс сотрудничества между разработчиками. Все файлы, находящиеся под версионным контролем, хранятся на центральном сервере. Совокупность клиентов взаимодействуют с сервером, работая с локальной копией необходимого файла.

Основные достоинства централизованной СКВ следующие:

- все знают, кто и чем занимается в проекте;

- администраторы осуществляют контроль над тем, кто и что может делать.

Главным недостатком централизованной СКВ является то, что централизованный сервер является уязвимым местом всей системы.

Распределенные системы контроля версий. В таких системах клиенты полностью копируют весь репозиторий.

Основные достоинства распределенной СКВ следующие:

- любой клиентский репозиторий может быть скопирован – возвращен на сервер для восстановления базы данных;
- возможность работы с несколькими удаленными репозиториями – можно одновременно работать по-разному с разными группами людей в рамках одного проекта.

Система управления версиями Git. Git – это гибкая, распределенная (без единого сервера) система контроля версий, дающая массу возможностей не только разработчикам программных продуктов, но и писателям для изменения, дополнения и отслеживания изменения «рукописей» и сюжетных линий, и учителям для корректировки и развития курса лекций, и администраторам для ведения документации, и для многих других направлений, требующих управления историей изменений. У каждого разработчика, использующего Git, есть свой локальный репозиторий, позволяющий локально управлять версиями. Затем сохраненными в локальном репозитории данными можно обмениваться с другими пользователями. Часто при работе с Git создают *центральный репозиторий*, с которым остальные разработчики синхронизируются.

Пример организации системы с центральным репозиторием – это проект разработки ядра Linux (<http://www.kernel.org>). В этом случае все участники проекта ведут свои локальные разработки и беспрепятственно скачивают обновления из центрального репозитория. Когда необходимые работы отдельными участниками проекта выполнены и отлажены, они, после удостоверения владельцем центрального репозитория в корректности и актуальности проделанной работы, загружают свои изменения в центральный репозиторий.

Наличие локальных репозиториях также значительно повышает надежность хранения данных, так как, если один из репозиториях выйдет из строя, данные могут быть легко восстановлены из других репозиториях.

Работа над версиями проекта в Git может вестись в нескольких ветках, которые затем с легкостью полностью или частично объединяются, уничтожаются, откатываются или разрастаются во все новые и новые ветки проекта.

Достоинства Git следующие:

- надежная система сравнения ревизий и проверки корректности данных, основанные на алгоритме хеширования SHA1 (Secure Hash Algorithm 1);
- гибкая система ветвления проектов и слияния веток между собой;
- наличие локального репозитория, содержащего полную информацию обо всех изменениях, что позволяет вести полноценный локальный контроль версий и заливать в главный репозиторий только полностью прошедшие проверку изменения;
- высокая производительность и скорость работы;
- удобный и интуитивно понятный набор команд;
- множество графических оболочек, позволяющих быстро и качественно вести работы с Git;
- возможность делать контрольные точки, в которых данные сохраняются без использования дельта-компрессии, т.е. полностью. Это позволяет уменьшить скорость восстановления данных, так как за основу берется ближайшая контрольная точка и восстановление идет от нее. Если бы контрольные точки отсутствовали, то восстановление больших проектов могло бы занимать часы;
- широкая распространенность, легкая доступность и качественная документация;
- гибкость системы, позволяющая удобно ее настраивать и даже создавать специализированные системы контроля или пользовательские интерфейсы на базе Git;
- универсальный сетевой доступ с использованием протоколов http, ftp, rsync, ssh и др.

Недостатки Git следующие:

- возможно (чрезвычайно редко) совпадения хеш-кода отличных по содержанию ревизий;
- не отслеживается изменение отдельных файлов, а только всего проекта целиком, что может быть неудобно при работе с большими проектами, содержащими множество несвязных файлов;
- при начальном (первом) создании репозитория и синхронизации его с другими разработчиками потребуется достаточно длительное время для скачивания данных, особенно, если проект большой, так как требуется скопировать на локальный компьютер весь репозиторий.

Экспериментальная часть занятия

Полезные команды:

- Tab – автоматическое дополнение текста команды и ее параметров по начальным символам.

- «стрелка вверх» – в рамках текущей сессии до закрытия консоли сохраняет историю команд.

Условие выполнения команд системы контроля версий – только с использованием консольной утилиты Git.

1. Подготовьте заранее рабочий каталог для работы с проектом и запомните путь.

2. Запустите консольную утилиту Git, в ее рабочем окне перейдите в заранее подготовленный каталог проекта с помощью команды **cd**:

```
cd d:\project
```

3. Выполните команду подготовки каталога проекта для работы с репозиторием:

```
git init
```

Проверьте создание подкаталога `.git` – основу Git-репозитория.

4. Основным инструментом для определения, какие файлы в каком состоянии находятся, является команда `git status`. Выполните эту команду и оцените результат:

- *On branch master* («В ветке master») – команда сообщает, на какой ветке (*branch*) проекта вы сейчас находитесь. Пока что это ветка *master*.

- *Initial commit* – у вас чистый рабочий каталог, в нем нет отслеживаемых измененных файлов.

- *Nothing to commit...* – нет неотслеживаемых файлов.

5. Создайте файл `readme.txt` в каталоге проекта, выполните команду

```
git status
```

На данный момент новый файл является «неотслеживаемым» (*untracked*). Это означает:

- файл виден, но отсутствует в предыдущем состоянии (коммите);
- файл не будет добавлен в новые коммиты, пока не будет явного указания.

Это предохранит вас от добавления в репозиторий случайных файлов.

6. Для добавления под версионный контроль необходимо проиндексировать эти файлы и осуществить первую фиксацию изменений. Выполните команду `git add`, указав имя индексируемого файла:

```
git add readme.txt
```

Варианты команды `git add`:

```
git add *.txt – добавить все текстовые файлы .txt
```

```
git add some_cat – добавить каталог some_cat
```

```
git add . – добавить все файлы текущего каталога.
```

7. Выполните команду `git status`, оцените сообщение. Файл `readme.txt` теперь отслеживаемый и индексированный, он находится в секции «Changes to be committed».

8. Зафиксируем наши изменения с помощью команды `git commit`. Каждый коммит обязательно должен быть сопровождается комментарием. Без аргументов откроется окно текстового редактора, где будет предложено набрать комментарий. В системе Linux/Unix необходимо знать команды редактора `vi` (`vim`). Также комментарий можно указать в командной строке:

```
git commit -m "Add readme file"
```

9. Создайте несколько `html`-страниц в рабочий каталог проекта. Добавьте все `html`-файлы в репозиторий и зафиксируйте изменения:

```
git add *.html
git commit -m "add all html files"
```

10. После создания нескольких снимков состояний может возникнуть необходимость просмотреть историю изменений проекта. Выполните команду `git log`. По умолчанию выводится список состояний в репозитории в обратном хронологическом порядке. Параметры команды представлены в табл. 20.1.

Таблица 20.1

Параметры команды `git log`

Параметр	Описание
<code>-p</code>	Показывает дельту (diff) для каждого состояния, опция -2 ограничивает вывод до двух последних записей: <code>git log -p -2</code>
<code>--since</code>	Список коммитов за указанных срок (пример, 2 недели): <code>git log --since=2.weeks</code>

11. Измените файл `file.txt`, выполните команду `git add` и `git commit`.

Замечание. Если вы изменили файл после выполнения `git add`, то вам придется снова выполнить `git add`, чтобы проиндексировать последнюю версию файла перед выполнением снимка состояния.

12. Если снимок состояния выполнен слишком рано, например нет добавления нужных файлов, то требуется сделать снимок повторно с опцией `--amend`.

Измените файл `forgotten_file.txt`. Выполните снимок состояния, выполните индексирование измененного файла, выполните повторный снимок состояния:

```
git commit -m 'initial commit'
git add forgotten_file
git commit -amend
```

13. Внесите изменения в два файла проекта, выполните их индексирование по команде `git add *`. Отмените индексацию одного из двух файлов на примере команды

```
git reset HEAD file_for_second_commit.txt
```

14. Измените один из файлов проекта, выполните возврат в исходное состояние на примере команды:

```
git checkout -- index.html
```

15. Для просмотра более результативного состояния файла, чем команда `git status` – используйте команду `git diff`. Эта команда отвечает на вопросы:

- 1) Что вы изменили, но еще не проиндексировали?
- 2) Что вы проиндексировали и собираетесь фиксировать?

Команда `git diff` показывает непосредственно добавленные и удаленные строки в виде заплатки (*patch*).

Наберите команду без аргументов:

```
git diff
```

16. Если вы хотите посмотреть, что вы проиндексировали и что войдет в следующий снимок состояния, выполните команду `git diff --cached`. В Git версии 1.6.1 и выше можно использовать `git diff --staged`. Эта команда сравнивает ваши индексированные изменения с последним снимком.

Команда `git diff` не показывает все изменения, сделанные с последнего коммита, а только не проиндексированные изменения. Вы можете использовать `git diff` для просмотра индексированных изменений (`git diff --cached`) в этом файле и не проиндексированных изменений (`git diff`).

Откройте файл `about.html` и измените его. Выполните команду `git diff`.

Оцените результат: красным цветом и знаком «-» отмечаются старые версии строк, а зеленым и знаком «+» – новые.

17. Выполните команду `git diff --staged`.

Мы видим пустой вывод, потому что еще не добавили файл `about.html` в индексируемые файлы, а команда `git diff --staged` сравнивает ваши индексированные изменения (то, что добавлено с помощью команды `git add`) с последним состоянием.

18. Выполните команды и оцените результат:

```
git add about.html
git diff --staged
```

19. Выполните команду и сравните результат с предыдущей командой:

```
git diff
```

20. Выполните удаление файла из вашего репозитория с удалением файла из вашего рабочего каталога помощью команд:

```
git rm labs.html
git status
git commit -m "delete labs"
```

21. Чтобы удалить файл из индекса, оставив его при этом в рабочем каталоге, используйте опцию `--cached`. Выполните команды и проверьте нахождение файла в статусе неотслеживаемых файлов:

```
git rm --cached readme.txt
git commit -m "delete readme from git index"
git status
```

22. Выполните перемещение файлов с помощью команды:

```
git mv Какой_файл Куда
```

Создайте папку «txt» и файл «file1.txt».

Например, чтобы переместить «file1.txt» в папку «txt», надо выполнить следующие команды, проверяя статус после каждого действия:

```
git mv file.txt txt/
git status
git commit -m 'move file1.txt'
git status
```

Контрольные вопросы

1. Какое назначение имеет программа GIT?
2. Что такое репозиторий?
3. Что такое ревизия?
4. Как просмотреть содержимое репозитория?
5. Как добавить файл, каталог в репозиторий?
6. Как обнаружить расхождения между репозиторием и его локальной копией?
7. Как откатиться к предыдущей версии файла?

СПИСОК ЛИТЕРАТУРЫ

1. *Зажигалкин А.В.* Стандартизация в оборонной промышленности. URL:<http://federalbook.ru/files/OPK/Soderjanie/OPK-7/III/Zajigalkin.pdf>.
2. Отчет о НИР «Контроль-ПО» 2М № 1514.
3. *Марковский А.С., Самонов А.В., Бурова И.О.* Организация автоматизированного контроля качества в жизненном цикле программных средств критически важных систем // Интеллектуальные технологии на транспорте.– 2016. – №1.
4. *Марковский А.С., Самонов А.В., Свеколкин Н. И.* Место и роль процессов контроля качества в жизненном цикле программных средств систем вооружения // Тр. ВКА им. А.Ф. Можайского. – СПб.: ВКА им. А.Ф. Можайского. – 2016. – №2.
5. Требования законодательства РФ по защите информации на предприятии: учеб. пособие / Е.Г. Воробьев, С.В. Войцеховский, Р.Г. Гильванов, А.С. Марковский. – М.: Микроинформ, 2005.
6. *Соммервиль Иан.* Инженерия программного обеспечения: пер. с англ. – 6-е изд. – М.: Издательский дом «Вильямс», 2002. – 624 с.
7. *Якобсон А., Буч Г., Рамбо Дж.* Унифицированный процесс разработки программного обеспечения. – СПб.: Питер, 2002. – 496 с.
8. *Константайн Л., Локвуд Л.* Разработка программного обеспечения. – СПб.: Питер, 2004. – 592 с.
9. *Иванова Г.С.* Технология программирования: учеб. для вузов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. – 320 с.
10. *Лохвицкий В.А.* Технологии разработки программного обеспечения: учеб. пособие. – СПб.: ВКА имени А.Ф. Можайского, 2014. – 123 с.
11. *Зиглер К.* Методы проектирования программных систем. – М.: Мир, 1985.
12. *Маклаков С.В.* Создание информационных систем с AllFusion Modeling Suite. – М.: ДИАЛОГ-МИФИ, 2003.
13. *Липаев В.В.* Качество программного обеспечения. – М.: Финансы и статистика, 1983. – С. 18–30.
14. *Майерс Г.* Надежность программного обеспечения. – М.: Мир, 1980.
15. CASE: Компьютерное проектирование программного обеспечения. – Изд-во Московского ун-та, 1994.
16. ГОСТ 34.602-89. Техническое задание на создание автоматизированной системы. – М.: Изд-во стандартов, 1989.
17. ГОСТ 19.201-78. Техническое задание. Требования к содержанию и оформлению. – М.: Изд-во стандартов, 1978.
18. ГОСТ Р. Порядок разработки программных средств систем вооружения ГОСТ Р 51189-98. – М.: Госстандарт России, 1998.
19. *Гусеница Я. Н., Петрич Д. О., Калиниченко С.В.* Информационная модель нарушения устойчивости функционирования автоматизированной системы // Защита информации. Инсайд. – 2016. – № 6. – С. 2–5.
20. *Сухов А. М., Якунин В. И., Калиниченко С.В.* Алгоритм применения методов и моделей противодействия компьютерным вторжениям // Защита информации. Инсайд. – 2016. – № 6. – С. 17–22.