ПРИОРИТЕТНЫЙ НАЦИОНАЛЬНЫЙ ПРОЕКТ «ОБРАЗОВАНИЕ»
РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Д. С. Кулябов, А. В. Королькова

Введение в формальные методы описания бизнес-процессов

Учебное пособие

Москва
2008

УДК 004.4’22, 658.5.012

Рецензент:
доктор технических наук, профессор О. Н. Ромашкова

Кулябов Д. С., Королькова А. В.
Введение в формальные методы описания бизнес-процессов: Учеб. пособие. — М.: РУДН, 2008. — 173 с.: ил.

В пособии рассматриваются основные нотации, применяемые при описании бизнес-процессов: нотации семейства IDEF, UML, BPMN. Демонстрируется взаимная связь разных нотаций. Даются примеры и рекомендации по использованию нотаций.

Для студентов направлений 550200 «Автоматизация и управление», 511200 «Математика, прикладная математика», 510400 «Физика», 521500 «Менеджмент», 521600 «Экономика», 060800 «Экономика и управление на предприятии (по отраслям производства)».

Оглавление

Введение

Основное назначение средств бизнес-моделирования — обеспечение понимания функционирования бизнес-процессов компании на всех уровнях организации. Бизнес-модель даёт целостную картину жизнедеятельности организации, согласовывает разные точки зрения на постоянно развивающуюся деятельность компании. Для наглядной демонстрации бизнес-процессов компании, анализа её архитектуры в целом и принятия решений об оптимизации её деятельности имеются специальные методики и языки моделирования.

Существующие средства проектирования бизнес-процессов применяют один из следующих подходов:

1) использование одной методологии и одной нотации проектирования для решения некоторой задачи (UML);
2) использование одной нотации для отображения нескольких разных точек зрения на одну проблему (BPMN);
3) использование нескольких методологий и нескольких нотаций для решения некоторой общей задачи (IDEF, ARIS).

В данном пособии даётся обзор базовых нотаций, используемых при бизнес-проектировании, — IDEF, UML, BPMN.

Семейство IDEF состоит из методологий, у каждой из которых есть чёткие границы применения и краткая, лаконичная нотация. Причём методологии в своей основе имеют разные парадигмы. Так, например, SADT/IDEF0, дополненные IDEF3 и DFD, а также IDEF1 представляют собой структурный подход к построению бизнес-модели. В то время как IDEF2 использует аппарат имитационного моделирования, а IDEF4 — представляет объектный подход к проектированию. При этом возникает проблема совмещения этих методологий. Решением является выбор точки зрения рассмотрения задачи. Каждый аналитик смотрит на проблему со своей точки зрения и использует свою нотацию. Общее представление о всех нотациях должен иметь руководитель

проекта, а аналитики ограничиваются знанием только своей области.

BPMN использует единую нотацию для отображения разных точек зрения на бизнес-процессы компании. При этом данная нотация в общем сложнее каждой отдельно взятой нотации IDEF.

В случае UML сделана попытка разработки единой нотации и единой методологии описания и моделирования бизнес-процессов. Создание UML стало попыткой заменить все остальные объектные парадигмы и выработать унифицированный метод построения бизнес-моделей. Однако для бизнес-аналитиков данный подход достаточно сложен, так как требует отказа от хорошо известных и привычных процедурных подходов и перехода к объектно-ориентированному мышлению. Кроме того, UML преподносится как язык моделирования общего назначения, который пытается достигнуть совместимости со всеми возможными языками разработки, что требует от аналитика знания не только нотации в целом, но и развитых навыков программирования. Поэтому UML со временем плавно переместился от общих задач бизнес-проектирования в сторону разработки программных систем. Следует отметить также, что нотация UML имеет тенденцию к неограниченному усложнению.

Учебное пособие предназначено для студентов, обучающихся по программе дополнительного образования «Информационно-телекоммуникационные системы». В рамках инновационной образовательной программы, реализованной в РУДН в 2008–2009 гг. на кафедре систем телекоммуникаций, разработан одноимённый учебно-методический комплекс (УМК), в состав которого входит электронный учебник. Программа дополнительного образования является авторской и включает в себя набор последовательно взаимоувязанных специальных дисциплин.

На программе могут обучаться студенты, не имеющие специального образования, например, обучающиеся по направлениям «Автоматизация и управление», «Математика, прикладная математика», «Физика», «Менеджмент», «Экономика», «Экономика и управление на предприятии (по отраслям производства)». Курс является составляющей модуля программы дополнительной

профессиональной подготовки «Основы управления инфокоммуникациями», которая включает также курсы: «Архитектура и принципы построения современных сетей и систем телекоммуникаций», «Введение в управление инфокоммуникациями», «Корпоративные информационные системы».

Первая глава знакомит читателя с базовыми понятиями управления бизнес-процессами и в области формальных языков описания бизнес-процессов. В данной теме вводятся основные понятия предметной области: бизнес-процесс, проектирование бизнес-процесса и т. п.

Во второй главе приводится краткий обзор поколений средств бизнес-моделирования.

В следующих четырёх главах подробно рассматривается нотация IDEF. Хотя IDEF принадлежит к первому поколению средств моделирования бизнес-процессов, данное средство является при этом востребованным в сфере моделирования бизнес-процессов. Так, например, IDEF0 может применяться при проектировании жизненного цикла бизнес-процесса, т. е. для создания функциональной модели, которая является структурированным отображением функций производственной системы или среды, а также информации и объектов, связывающих эти функции. IDEF1 тесно связан с проектированием, разработкой, эксплуатацией баз данных, т. е. применяется для построения информационной модели, которая представляет структуру информации, необходимую для поддержки функций производственной системы или среды. IDEF2 позволяет построить динамическую модель меняющегося во времени поведения функций, информации и ресурсов производственной системы или среды. IDEF3 используется для разработки диаграмм перехода состояний и автоматного проектирования, т. е. для сбора информации о состоянии моделируемой системы, и представляет собой структурный метод, показывающий причинно-следственные связи и события. IDEF4 расширяет структурную модель объектно-ориентированными средствами.

В седьмой главе рассматриваются диаграммы потоков данных DFD, которые показывают, как обрабатывает информацию каждый процесс, а также

демонстрируют взаимодействие процессов друг с другом. Методика DFD рассматривается как дополнительное средство моделирования к IDEF0 наряду с IDEF3.

Восьмая глава достаточно подробно рассматривает концепцию UML как языка моделирования сложных систем. UML привнёс в бизнес-проектирование новую парадигму — структурный подход IDEF был полностью заменён на объектно-ориентированный подход.

В девятой главе рассматривается концепция моделирования бизнес-процессов на основе нотации BPMN и языка моделирования BPML как средства моделирования третьего поколения, позволяющего создавать новые процессы «налету». В данной теме изучаются основные элементы языка BPML, при этом BPMN рассматривается как графическая нотация языка BPML. Описываются основные элементы этой нотации.

В последней главе приводится обзор других технологий проектирования бизнес-процессов.

В списке источников даны ссылки на нормативные документы, статьи и монографии, которые использовались при написании основного текста пособия. Список рекомендуемой литературы содержит учебную литературу по рассматриваемой тематике.

Глава 1. Базовые понятия в области управления бизнес-процессами и в области формальных языков описания бизнес-процессов

1.1. Понятие бизнес-процесса

Бизнес-процесс определяется как логически завершённый набор взаимосвязанных и взаимодействующих видов деятельности, поддерживающий деятельность организации и реализующий её политику, направленную на достижение поставленных целей [1, 2].

Бизнес-модель определяется как формализованное (графическое, табличное, текстовое, символьное) описание бизнес-процессов, отражающее реально существующую или предполагаемую деятельность предприятия [1].

Моделирование бизнес-процессов включает следующие цели:

— обеспечение понимания структуры организации и динамики происходящих в ней процессов;
— обеспечение понимания текущих проблем организации и возможностей их решения;
— обеспечение единого восприятия заказчиками, пользователями и разработчиками целей и задач организации;
— создание основы для формирования требований к программному обеспечению, автоматизирующему бизнес-процессы организации.

Модель бизнес-процесса должна определять:

— процедуры (функции, работы), которые необходимо выполнить для получения заданного конечного результата;
— последовательность выполнения процедур;
— механизмы контроля и управления в рамках рассматриваемого бизнес-процесса;

— субъекты выполнения процедур процесса;
— входящие документы / информацию, используемые каждой процедурой процесса;
— исходящие документы / информацию, генерируемые процедурами процесса;
— ресурсы, требующиеся для выполнения каждой процедуры процесса;
— документацию / условия, регламентирующие выполнение процедуры;
— параметры, характеризующие выполнение процедур и процесса в целом.

1.2. Подход к моделированию бизнес-процессов

При описании деятельности той или иной организации необходимо помнить, что крайне важным моментом является постановка и формализация цели описания.

Своеобразным шаблоном при моделировании деятельности организации является *графическое представление*, которое отражает поведение компании как системы, позволяя разложить стратегические цели компании на отдельные составляющие и довести их до конечных исполнителей. При этом вся деятельность разбивается на три уровня:

1) уровень целей;
2) уровень окружающей среды;
3) уровень внутренней организации.

Процессы первого уровня формулируют проблемную ситуацию, которую решает организация, и ставят глобальную цель для всей компании, в результате чего формируются:

— миссия, видение и философия организации;
— направление развития бизнеса;
— ключевые показатели, релевантные для оценки достижения глобальной цели (рост, прибыльность, эффективность и пр.).

Процессы второго уровня координируются результатами процессов первого уровня и связывают текущую деятельность подразделений и исполнителей с формализованными целями организации, в результате чего осуществляется:

— выявление угроз, возможностей, слабых и сильных сторон;
— стратегический анализ и выработка стратегических альтернатив деятельности компании;
— выбор способов достижения поставленной глобальной цели для формирования конкурентоспособного предложения на рынке;
— определение точек контроля, нормативных значений параметров, характеризующих качество выполнения последующих процессов текущей операционной деятельности.

На третьем уровне выполняются следующие действия:

— выявляются и формируются элементы организации;
— определяются отношения между элементами, реализующие целенаправленное функционирование организации;
— выбираются способы реализации связей между элементами;
— множество образованных связей и отношений между элементами упорядочиваются в структуру организации, а характеристики выбранных способов связей и основные требования к функционированию элементов являются требованиями к информационной системе, которая будет осуществлять данную связь;
— проектируются процессы текущей операционной деятельности, отчётность, документооборот.

В результате такого моделирования строятся управляющие процессы, являющиеся ортогональными к процессам оперативной деятельности. Переход от уровня к уровню понимается именно как процесс, а на графической схеме получается два ортогональных направления процессов: одно (горизонтальное) — оперативное, второе (вертикальное) — управления. Причём информацией для вертикального процесса, кроме внешней информации, является

результат процессов последующих уровней, в то время как результат процесса предыдущего уровня является управляющей информацией для процессов последующего уровня. Таким образом реализуется управляющая обратная связь.

1.3. Базовые понятия в области формальных языков описания бизнес-процессов

1.3.1. Системы управления бизнес-процессами

Компьютерные системы, основанные на процессном подходе к управлению бизнес-системами, получили название *систем управления бизнес-процессами или BPM-систем (Business Process Management)*[1].

BPM-система должна выполнять две основные роли:

— формировать единый язык описания управления бизнес-процессами;
— обеспечивать быструю интеграцию в рамках единого процесса деятельности сотрудников и компьютерных систем предприятия.

1.3.2. Математические основы языков описания бизнес-процессов

В основе большинства языков описания бизнес-процессов лежит одна из двух математических теорий:

— теория сетей Петри;
— концепция «π-исчисления» («Pi calculus»).

Теория сетей Петри основана на классической теории графов. Теория включает графическую нотацию — систему графических обозначений, на основе которых можно строить соответствующие графы. Считается, что использование для описания BPM-систем концепции сетей Петри в явном виде неудобно.

[1] Многие термины в области управления бизнес-процессами ещё не устоялись. Поэтому в литературе можно встретить разные названия одинаковых по сути терминов. Так, например, WorkFlow-системы, DocFlow-системы являются частными случаями BPM-систем и т.п.

Тем не менее ряд формальных языков описания бизнес-процессов (например, WPDL и XPDLкоалиции WfMC) включают в себя многие понятия и концепции сетей Петри, такие как узлы, переходы, условия и т.д.

Концепция π-исчисления была разработана в конце 80-х г. XX в. Робином Милнером и основана на алгебре параллельных процессов. В отличие от сетей Петри, математическими объектами π-исчисления являются не графы, а выражения над элементами специальных множеств и преобразования над этими выражениями. К формальным языкам описания бизнес-процессов на базе π-исчисления относят BPML и BPEL.

1.3.3. WorkFlow- и DocFlow-системы

BPM-системы делятся на *WorkFlow-системы* и *DocFlow-системы*.

Основным элементом WorkFlow-системы является *поток работ*. В WorkFlow-системе деятельность бизнес-системы можно представить в виде элементов работы, перемещающихся по определённому маршруту между исполнителями в соответствии с заданными правилами. При этом от одного исполнителя к другому передаётся точка управления. Данная парадигма легко представима в виде графа.

Основным элементом DocFlow-системы является *поток документов*. В DocFlow-системе деятельность бизнес-системы можно представить в виде документов, перемещающихся между их редакторами по определённому маршруту в соответствии с заданными правилами. При этом от одного редактора к другому передаётся не точка управления, а «корзины» документов.

В настоящее время WorkFlow- и DocFlow-системы представляют собой системы разных типов, однако постепенно системы документооборота по функциональности приближаются к WorkFlow-системам. При помощи DocFlow-систем можно моделировать многие виды бизнес-процессов, а посредством WorkFlow-систем — автоматизировать элементы документооборота.

1.3.4. Перспектива бизнес-процесса

Назовём *перспективой бизнес-процесса* точку зрения или слои/уровни рассмотрения бизнес-процесса.

Выделяют следующие перспективы:

— перспектива управления потоком (Control-Flow Perspective);
— перспектива данных (Data Perspective);
— перспектива ресурсов (Resource Perspective);
— перспектива операций (Operational Perspective).

Перспектива управления потоком представляет собой граф со множеством узлов, соединённых между собой переходами, по которым согласно некоторым правилам перемещается точка управления (указатель на активный узел процесса).

В узле, соответствующем некоторому шагу процесса, система даёт задание (работу) исполнителю (сотруднику или информационной системе) и ожидает ответа (сообщения) о выполнении работы. После поступления сообщения о выполнении работы точка управления перемещается по переходу к следующему узлу процесса. При этом к узлу, соответствующему шагу процесса, может примыкать только один входящий и один исходящий переход.

Маршрутный узел соответствует разветвлению или слиянию точек управления, где система на основании некоторых правил выбирает следующий узел, в который будет передано управление. С маршрутными узлами обязательно связано более одного входящего или исходящего перехода.

Перспектива данных представляет собой набор переменных и данных внешних информационных систем, которые используются при исполнении бизнес-процесса.

Переменные бизнес-процесса могут являться входящими и исходящими параметрами при взаимодействии системы с информационными системами предприятия. При помощи внутренних переменных происходит обмен информацией между шагами процесса и, как следствие, между внешними ин-

формационными системами, а также выбор конкретного внутреннего перемещения точки управления между узлами по одному из возможных переходов. Выбор перехода осуществляется согласно правилам бизнес-логики процесса, описанной в перспективе управления потоком.

Перспектива ресурсов представляет собой список исполнителей, которые могут выполнять шаги бизнес-процесса. Исполнителями могут быть как сотрудники предприятия, так и информационные системы или специализированные устройства.

Перспектива операций представляет собой список элементарных действий, совершаемых исполнителями в рамках шага бизнес-процесса.

Глава 2. Поколения средств моделирования бизнес-процессов

2.1. Методы моделирования бизнес-процессов

В основе методов моделирования бизнес-процессов могут лежать как структурный, так и объектно-ориентированный подходы к моделированию. Перечислим некоторые из методов:

- метод функционального моделирования SADT/IDEF0;
- метод моделирования процессов IDEF3;
- моделирование потоков данных DFD;
- нотация моделирования потоков работ BPMN;
- метод ARIS;
- метод моделирования, используемый в технологии Rational Unified Process.

2.2. SADT

Методология структурного анализа и проектирования (Structured Analysis and Design Technique, SADT) была создана в конце 60-х гг. XX в. Дугласом Россом. SADT нашла своё применение в области описания большого количества сложных искусственных систем из широкого спектра областей. В 1973 г. впервые при помощи SADT был реализован крупный аэрокосмический проект. В виде конечного продукта SADT появилась в 1975 г. К 1981 г. методология SADT использовалась более чем в 50 компаниях при работе над проектами, охватывавшими различные проблемные области, в том числе телефонные сети, аэрокосмическое производство, управление и контроль, учёт материально-технических ресурсов. Причина успешного и разнообразного её

применения заключается в том, что SADT является полной методологией для создания описания систем, основанной на концепциях системного моделирования.

Метод SADT поддерживается Министерством обороны США, которое было инициатором разработки семейства стандартов *IDEF (ICAM DEFinition)*, являющегося основной частью программы *ICAM (Integrated Computer Aided Manufacturing — интегрированная компьютеризация производства)*, проводимой по инициативе военно-воздушных сил США. Метод SADT реализован в одном из стандартов этого семейства — IDEF0, который был утверждён в качестве федерального стандарта США в 1993 г. [3].

2.3. IDEF

Методологии семейства IDEF позволяют отображать и анализировать модели деятельности широкого спектра сложных систем в различных разрезах.

В настоящий момент в семейство IDEF входят:

- IDEF0 — методология функционального моделирования (изучаемая система представляется в виде набора взаимосвязанных функций — функциональных блоков);
- IDEF1 — методология моделирования информационных потоков внутри системы, позволяющая отображать и анализировать их структуру и взаимосвязи;
- IDEF1X (IDEF1 eXtended) — методология построения реляционных структур (как правило, используется для моделирования реляционных баз данных, имеющих отношение к рассматриваемой системе);
- IDEF2 — методология динамического моделирования развития систем;
- IDEF3 — методология документирования процессов, происходящих в системе;
- IDEF4 — методология построения объектно-ориентированных систем, позволяющая наглядно отображать структуру объектов и заложенные

принципы их взаимодействия;

— IDEF5 — методология онтологического исследования сложных систем при помощи определённого словаря терминов и правил, на основании которых могут быть сформированы достоверные утверждения о состоянии рассматриваемой системы в некоторый момент времени;
— IDEF6 — методология использования рационального опыта проектирования, позволяющая предотвратить возникновение структурных ошибок при новом проектировании информационных систем;
— IDEF7 — методология аудита информационной системы;
— IDEF8 — методология разработки модели графического интерфейса пользователя;
— IDEF9 — методология анализа существующих условий и ограничений, их влияния на принимаемые решения в процессе реинжиниринга;
— IDEF10 — методология моделирования архитектуры выполнения;
— IDEF11 — методология информационного моделирования артефактов;
— IDEF12 — методология организационного моделирования;
— IDEF13 — методология проектирования трёхсхемного дизайна карт;
— IDEF14 — методология моделирования компьютерных сетей.

2.3.1. IDEF0

Как сказано выше, стандарт IDEF0 был разработан в 1981 г. департаментом ВВС США в рамках программы ICAM. Методология IDEF0 представляет собой развитие графического языка описания функциональных систем SADT. Метод должен был обеспечить групповую работу над созданием модели, с непосредственным участием всех аналитиков и специалистов, занятых в рамках проекта.

IDEF0 реализует методику функционального моделирования сложных систем. Применять IDEF0 рекомендуется для начальных стадий проектирова-

ния сложных искусственных систем управления, производства, бизнеса, включающих людей, оборудование, программное обеспечение.

Подробнее методология IDEF0 будет рассмотрена в главе 3.

2.3.2. IDEF1 и IDEF1X

IDEF1 и IDEF1X реализуют методики инфологического проектирования баз данных.

Стандарт IDEF1 был разработан как инструмент для анализа и изучения взаимосвязей между информационными потоками анализируемой системы. Целью подобного исследования является структуризация и дополнение существующей информации, обеспечение качественного управления информационными потоками.

Применение методологии IDEF1 позволяет решить следующие задачи:

— выяснить структуру и содержание существующих потоков информации в системе;

— выявить проблемы, вызванные недостатком управления соответствующими потоками информации;

— выявить информационные потоки, требующие дополнительного управления для эффективной реализации модели.

IDEF1 является аналитическим методом и используется преимущественно для выполнения следующих действий:

— определение информации, имеющей отношение к деятельности системы, а также структуры её потоков;

— определение существующих правил и законов, по которым осуществляется движение информационных потоков, а также принципов управления ими;

— выяснение взаимосвязей между существующими информационными потоками в рамках системы;

— выявление проблем, возникающих при некачественном информационном управлении.

IDEF1X является методом для разработки реляционных баз данных. Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных после исследования другими методами информационных ресурсов системы и принятия решения о внедрении реляционной базы данных, как части корпоративной информационной системы. Основным преимуществом IDEF1X, по сравнению с другими методами разработки реляционных баз данных, является жёсткая и строгая стандартизация моделирования, которая позволяет избежать различной трактовки построенной модели.

Подробнее методология IDEF0 будет рассмотрена в главе 4.

2.3.3. IDEF2 и IDEF3

Методологии IDEF2 и IDEF3 [4] реализуют поведенческое моделирование системы. Если методика IDEF0 связана с функциональными аспектами и позволяет отвечать на вопрос: «Что делает эта система?», то в этих методиках детализируется ответ на вопрос: «Как система это делает?». В основе поведенческого моделирования лежат модели и методы имитационного моделирования систем массового обслуживания, сети Петри, возможно применение модели конечного автомата, описывающей поведение системы как последовательности смен состояний.

Методология IDEF3 разработана в конце 1980-х гг. для моделирования последовательности действий и процессов анализируемой системы. На данный момент методология IDEF3 является стандартом документирования процессов, происходящих в системе.

Основой методологии служит сценарий процесса, выделяющий из модели последовательность происходящих в системе действий. Диаграммы IDEF3 позволяют моделировать сценарии происходящих в системе процессов, исследовать связи между действиями процессов, описывать последовательно-

сти изменений свойств объекта в рамках рассматриваемого процесса.

Средства моделирования и документирования IDEF3 позволяют выполнять следующие задачи:

— документировать имеющиеся данные о технологии процесса;
— определять и анализировать точки влияния потоков сопутствующего документооборота на сценарий технологических процессов;
— определять ситуации, в которых требуется принятие решения, влияющего на жизненный цикл процесса;
— содействовать принятию оптимальных решений при реорганизации технологических процессов;
— разрабатывать имитационные модели технологических процессов.

2.3.4. IDEF4

Методология IDEF4 [5] реализует объектно-ориентированный анализ больших систем. Методология предоставляет пользователю графический язык для изображения классов, диаграмм наследования, таксономии методов.

2.3.5. IDEF5

Методология IDEF5 [6] обеспечивает наглядное представление данных, полученных в результате обработки онтологических запросов в простой естественной графической форме.

Как схематический язык, IDEF5 ближе всего к IDEF1 и IDEF1X. Информация, отражающаяся в модели IDEF1 или IDEF1X, может быть выражена в IDEF5. Но для проектирования реляционных баз данных IDEF5 не подходит, так как не содержит хорошо продуманных, специализированных представлений IDEF1/1X.

2.3.6. IDEF6

Методология IDEF6 направлена на сохранение рационального опыта проектирования информационных систем, что способствует предотвращению структурных ошибок.

IDEF6 используется для описания и обоснования дизайна разрабатываемой информационной системы, а также для отображения связи проектных решений по разработке моделей и системы документации. Преимуществом данной методологии является то, что она помогает организации избежать повторения ошибок проектирования, определяя влияние вносимых в дизайн системы изменений.

В отличие от других методик IDEF, в которых фиксируются результаты проектирования, в IDEF6 главный упор сделан на пути получения этих результатов и обосновании промежуточных решений. Такой подход особенно важен при разработке сложных систем в недостаточно определённых ситуациях. Фиксация шагов и обоснований помогает при дальнейших модернизациях систем, сохранению и использованию рационального опыта проектирования. Методика упорядочивает обнаружение и устранение неопределённостей, ошибок, неудовлетворительных ограничений.

2.3.7. IDEF8

Методология IDEF8 предназначена для проектирования взаимодействия человека и пользовательской системы. Создаваемые сценарии должны удовлетворять ряду оговорённых в методике принципов, таких как уменьшение нагрузки на человека, идентичность средств диалога в разных системах, наличие обратной связи для исправления ошибок, хранение истории диалога, помощь советами по выполнению действий и т.д.

2.3.8. IDEF9

Методология IDEF9 [7] предназначена для анализа имеющихся условий и ограничений (в том числе физических, юридических, политических) и их влияния на принимаемые решения в процессе реинжиниринга. Данная методика призвана помочь в обнаружении и анализе проблем, возникающих в бизнес-системах. Выявление ограничений в бизнес-системе и их систематический анализ позволяют улучшить функционирование системы.

Обычно в качестве систем фигурируют сложные информационные системы с ориентацией на экономические и управленческие приложения. Под ограничением понимается отношение, которое должно соблюдаться. Ограничения делятся на контексты (группы родственных ограничений).

Применение IDEF9 заключается в выполнении нескольких шагов:

1) сбор свидетельств (фактов, указывающих на наличие ограничения);
2) классификация — определение контекстов, объектов, отношений;
3) прогнозирование — выявление ограничений на основе свидетельств;
4) отбор значимых ограничений;
5) определение экспертов для тестирования результатов;
6) детализация и фильтрация ограничений.

В методике даны рекомендации по выполнению этих шагов. Предлагается графический язык, элементами которого являются система, блоки ограничений, контексты, линии связи, логические связки OR, AND, XOR (исключающее ИЛИ).

2.3.9. IDEF14

Методология IDEF14 предназначена для представления и анализа данных при проектировании вычислительных сетей на графическом языке с описанием конфигураций, очередей, сетевых компонентов, требований к надёжности и т.п.

Чаще всего методика применяется для модернизации уже существующих сетей. Проектирование включает в себя определение топологии сети или схемы коммуникаций, реализацию нужного качества обслуживания, анализ функционирования (трафик, дисциплины обслуживания в узлах, протоколы доступа). Модель топологии дополняется моделями очередей, надёжности, материальных затрат. Важную роль играет библиотека методов построения и компонентов сетей. Методика основана на выполнении ряда шагов: установление целей модернизации, исследование существующей сети, определение типов компонентов в ней, построение модели «как есть», её верификация, анализ результатов, корректировка с переходом к модели «как должно быть» .

2.4. DFD

Диаграммы потоков данных (Data Flow Diagramming, DFD) представляют собой иерархию процессов, которые связаны между собой потоками данных. Диаграммы показывают, как обрабатывает информацию каждый процесс, как процессы связаны друг с другом, а также как работает сама система, каким образом она обрабатывает поступающие данные.

2.5. UML

Унифицированный язык моделирования (Unified Modeling Language, UML) представляет собой общецелевой язык визуального моделирования, который разработан для спецификации, визуализации, проектирования и документирования компонентов программного обеспечения, бизнес-процессов и других систем.

Язык UML является результатом совместной работы Г. Буча (G. Booch), Д. Рамбо (J. Rumbaugh), И. Якобсона (I. Jacobson) и др. Г. Буч разработал нотацию графических символов для описания различных аспектов модели, Д. Рамбо — нотацию технологии объектного моделирования (Object Mod-

eling Technology, OMT). И. Якобсон — впервые описал процесс выявления и фиксации требований к системе в виде совокупностей транзакций, а также разработал метод проектирования систем под названием «Объектно-ориентированное проектирование программного обеспечения» (Object Oriented Software Engineering, OOSE).

Процесс консолидации методов, впоследствии вошедших в UML, начался в 1993 г. В октябре 1995 г. была выпущена предварительная версия 0.8 унифицированного метода (Unified Method). Затем консорциум OMG (Object Management Group), образованный ещё в 1989, выпустил в 1996 г. предварительную версию спецификации UML. К разработке новых версий языка в рамках консорциума UML Partners присоединились такие компании, как Digital Equipment Corporation, Hewlett-Packard, i-Logix, IntelliCorp, IBM, ICON Computing, MCI Systemhouse, Microsoft, Oracle Corporation, Rational Software, Texas Instruments и Unisys. Результатом их совместной работы стала спецификация UML 1.0, вышедшая в январе 1997 года. Последующие релизы UML включали версии 1.3, 1.4 и 1.5, опубликованные, соответственно, в июне 1999 г., сентябре 2001 г. и марте 2003 г. Формальная спецификация последней версии UML 2.0 опубликована в августе 2005 г. Семантика языка была значительно уточнена и расширена для поддержки методологии Model Driven Development (MDD). UML 1.4.2 принят в качестве международного стандарта ISO/IEC 19501:2005 [8].

UML содержит в себе механизмы расширения, предназначенные для адаптации определённого языка моделирования к конкретным требованиям разработчика без необходимости изменения метамодели. Наличие механизмов расширения принципиально отличает UML от таких средств моделирования, как IDEF0, IDEF1X, IDEF3, DFD, которые сильно типизированы, т.к. не допускают произвольной интерпретации семантики элементов моделей. UML, допуская такую интерпретацию, является слабо типизированным языком.

Язык UML используется также в методе моделирования бизнес-процессов, являющемся частью технологии *Rational Unified Process (RUP)* компа-

нии IBM Rational Software. Этот метод, направленный прежде всего на создание основы для формирования требований к программному обеспечению, предусматривает построение двух базовых моделей: *модели бизнес-процессов (Business Use Case Model)* и *модели бизнес-анализа (Business Analysis Model)*.

2.6. BPMN, BPEL, BPML

Business Process Modeling Notation (BPMN) представляет собой графическую нотацию для отображения бизнес-процессов при моделировании потоков работ, происходящих в исследуемой системе.

Нотация BPMN была разработана организацией Business Process Management Initiative (BPMI), в настоящее время разработка BPMN ведётся консорциумом OMG (Object Management Group). Выходу первой версии BPMN в 2003–2004 гг. предшествовало появление в 2000 г. книги о *системах управления бизнес-процессами (Business Process Management Systems, BPMS)*, первой версии BPMS в 2001 г., спецификации BPML 1.0 в 2002 г. В 2005 г. стандарт был передан OMG, и в 2006 г. вышла спецификация BPMN 1.0 от OMG [9]. В 2007 г. OMG выпустила спецификацию BPMN 2.0 [10].

Целью проекта BPMN является создание общей нотации разработки моделей бизнес-процессов для различных категорий специалистов: от аналитиков и экспертов, моделирующих бизнес-процессы, технических разработчиков, которые создают системы для выполнения этих процессов, до менеджеров различных уровней, которые должны понимать процессные диаграммы, чтобы принимать деловые решения.

Благодаря абстрактному представлению модели нотация BPMN позволяет наглядным образом описывать модели бизнес-процессов независимо от среды их функционирования. Для реализации нотации модели используются языки исполнения бизнес-процессов — *BPML (Business Process Modeling Language)* и *BPEL (Business Process Execution Language)*.

Язык определения бизнес-процессов BPML, основанный на технологии Web-сервисов, разработан коалицией BPMI в 2002 г. В настоящее время есть возможность экспорта из графической нотации BPMN как в BPML, так и в BPEL.

Язык BPEL (или BPEL4WS) был разработан группой ИТ-компаний, где ведущую роль в разработке языка сыграли IBM и Microsoft. BPEL фактически стал прямым наследником ранее созданных спецификаций IBM WSFL и Microsoft XLANG. В нём используются сразу несколько XML-спецификаций — WSDL, XML Schema, XPath.

В марте 2003 г. BPEL был утверждён в качестве стандарта *организацией по продвижению стандартов в области структурированной информации (Organization for the Advancement of Structured Information Standards, OASIS)*. Весной 2004 г. появилась версия BPEL 1.1. Эта спецификация была впервые реализована в продуктах IBM WebSphere Business Integration Server Foundation 5.1 и Microsoft BizTalk Server 2004. Кроме того, поддержка BPEL обеспечивается в ряде ведущих платформ исполнения приложений и сервисов (например, SAP NetWeaver и BEA WebLogic).

BPEL представляет собой метаязык на базе XML, позволяющий определить последовательность выполнения функционала Web-сервисов в ходе различных потоков операций (транзакций).

Глава 3. Методика IDEF0/SADT. Функциональная модель

3.1. Методология SADT/IDEF0

SADT-моделью называется описание системы с помощью SADT. В SADT-моделях используются как *естественный*, так и *графический* языки. Естественный язык служит для передачи информации о конкретной системе. При этом источником естественного языка являются люди, описывающие систему. Графический язык SADT определённым образом организует естественный язык. Источником графического языка служит сама методология SADT.

С точки зрения SADT модель может быть сосредоточена либо на *функциях* системы, либо на её *объектах*. SADT-модели, ориентированные на функции, принято называть *функциональными моделями*, а ориентированные на объекты системы — *моделями данных*.

Функциональная модель представляет с требуемой степенью детализации систему функций, которые отражают свои взаимоотношения через объекты системы. *Модели данных* представляют собой подробное описание объектов системы, связанных системными функциями. Полная методология SADT поддерживает создание множества моделей для более точного описания сложной системы.

Методология SADT/IDEF0 представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями.

Основные элементы методологии SADT/IDEF0 основываются на следующих концепциях:

— графическое представление блочного моделирования — графика блоков и дуг SADT-диаграммы — отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него;
— строгость и точность — выполнение правил SADT требует достаточной строгости и точности, не накладывая в то же время чрезмерных ограничений на действия аналитика;
— связность диаграмм — блоки нумеруются специальным образом;
— уникальность меток и наименований — отсутствие повторяющихся имён;
— синтаксические правила для графических символов (блоков и стрелок);
— разделение входов и управлений — правило определения роли данных;
— отделение организации от функции, т.е. исключение влияния организационной структуры на функциональную модель.

Методология SADT/IDEF0 может использоваться для моделирования широкого круга систем и определения требований и функций, а затем для разработки системы, которая удовлетворяет этим требованиям и реализует эти функции. Для уже существующих систем SADT/IDEF0 может быть использована для анализа функций, выполняемых системой, а также для указания механизмов, посредством которых они осуществляются.

3.1.1. Методологические понятия

Одним из основных понятий стандарта IDEF0 является *декомпозиция (Decomposition)*. Принцип декомпозиции применяется при разбиении сложного процесса на составляющие его функции. При этом уровень детализации процесса определяется непосредственно разработчиком модели. Декомпозиция позволяет постепенно и структурированно представлять модель системы в виде иерархической структуры отдельных диаграмм.

Модель IDEF0 всегда начинается с представления системы как единого целого — одного функционального блока с интерфейсными дугами, уходящи-

ми за пределы рассматриваемой области. Такая диаграмма с одним функциональным блоком называется *контекстной диаграммой* и обозначается идентификатором «A-0».

В пояснительном тексте к контекстной диаграмме должна быть указана *цель (Purpose)* построения диаграммы в виде краткого описания и зафиксирована *точка зрения (Viewpoint)*. Фактически цель определяет соответствующие области в исследуемой системе, на которых необходимо фокусироваться в первую очередь. Точка зрения определяет основное направление развития модели и уровень необходимой детализации. Чёткое фиксирование точки зрения позволяет разгрузить модель, отказавшись от детализации и исследования отдельных элементов, не являющихся необходимыми, исходя из выбранной точки зрения на систему.

В процессе декомпозиции функциональный блок, который в контекстной диаграмме отображает систему как единое целое, подвергается детализации на другой диаграмме. Получившаяся диаграмма второго уровня содержит функциональные блоки, отображающие главные подфункции функционального блока контекстной диаграммы, и называется *дочерней (Child Diagram)* по отношению к нему, а каждый из функциональных блоков, принадлежащих дочерней диаграмме, соответственно называется *дочерним блоком (Child Box*). В свою очередь, функциональный блок-предок называется *родительским блоком по отношению к дочерней диаграмме (Parent Box)*, а диаграмма, к которой он принадлежит, — *родительской диаграммой (Parent Diagram)*. Каждая из подфункций дочерней диаграммы может быть далее детализирована путём аналогичной декомпозиции соответствующего ей функционального блока. В случае декомпозиции функционального блока все интерфейсные дуги, входящие в данный блок или исходящие из него, фиксируются на дочерней диаграмме.

Каждый блок имеет свой уникальный порядковый номер на диаграмме (цифра в правом нижнем углу прямоугольника), а обозначение под правым углом указывает на номер дочерней для этого блока диаграммы. Отсутствие

этого обозначения говорит о том, что декомпозиции для данного блока не существует.

Для каждого из элементов IDEF0 (диаграмм, функциональных блоков, интерфейсных дуг) существующий стандарт подразумевает создание и поддержание набора соответствующих определений, ключевых слов, повествовательных изложений и т.д., которые характеризуют объект, отображённый данным элементом. Этот набор называется *глоссарием (Glossary)* и является описанием сущности данного элемента.

3.1.2. Точка зрения

С методической точки зрения при моделировании полезно использовать мнения экспертов, имеющих разные взгляды на предметную область. Однако при разработке каждой модели должна существовать единственная, заранее определённая точка зрения.

Обычно наименованием точки зрения является название должности или подразделения.

Может оказаться необходимым построение моделей с разных точек зрения для детального отражения всех особенностей моделируемой системы. Тогда в прикреплённых диаграммах кратко документируют и другие точки зрения.

3.1.3. Иерархия диаграмм

Построение IDEF0-модели начинается с представления всей системы в виде простейшей компоненты — одного блока и стрелок, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок представляет всю систему как единое целое, то имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг — они также представляют полный набор внешних интерфейсов системы в целом.

Далее блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединён-

ных интерфейсными дугами. Эти блоки представляют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых представлена как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть разбита подобным образом для более детального представления.

Во всех случаях каждая подфункция может содержать только те элементы, которые входят в исходную функцию. Кроме того, модель не может опустить какие-либо элементы, так как родительский блок и его интерфейсы обеспечивают контекст. К нему нельзя ничего добавить, и из него не может быть ничего удалено.

Модель IDEF0 представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, которые представлены в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах. Каждая детальная диаграмма является декомпозицией блока из более общей диаграммы. На каждом шаге декомпозиции более общая диаграмма называется родительской для более детальной диаграммы.

Стрелки, входящие в блок и выходящие из него, на диаграмме верхнего уровня являются теми же, что и стрелки, входящие в диаграмму нижнего уровня и выходящие из неё, потому что блок и диаграмма представляют одну и ту же часть системы. Все граничные дуги должны продолжаться на родительской диаграмме, чтобы она была полной и непротиворечивой.

3.1.4. Принципы ограничения сложности IDEF0-диаграмм

Обычно IDEF0-модели несут в себе сложную и концентрированную информацию. Поэтому, чтобы ограничить их перегруженность и сделать удобочитаемыми, в стандарте приняты соответствующие ограничения сложности:

— ограничение количества функциональных блоков на каждом уровне декомпозиции тремя–шестью (верхний предел (шесть) заставляет разра-

ботчика использовать иерархии при описании сложных предметов, а нижний предел (три) гарантирует, что на соответствующей диаграмме достаточно деталей, чтобы оправдать её создание);

— ограничение количества подходящих к одному функциональному блоку (выходящих из одного функционального блока) интерфейсных дуг четырьмя.

3.1.5. Дисциплина групповой работы над разработкой IDEF0-модели

Стандарт IDEF0 содержит набор процедур, позволяющих разрабатывать и согласовывать модель большой группой людей, принадлежащих к разным областям деятельности моделируемой системы. Обычно процесс разработки является итеративным и состоит из следующих условных этапов.

— *Создание модели группой специалистов, относящихся к различным сферам деятельности предприятия, называемых авторами (Authors)*. Построение первоначальной модели является динамическим процессом, в течение которого авторы опрашивают компетентных лиц о структуре различных процессов. На основе имеющихся положений, документов и результатов опросов создаётся *черновик (Model Draft)* модели.

— *Распространение черновика для рассмотрения, согласований и комментариев.* На этой стадии происходит обсуждение черновика модели с широким спектром компетентных лиц (читателями). При этом каждая из диаграмм черновой модели письменно критикуется и комментируется, а затем передаётся автору. Автор в свою очередь также письменно соглашается с критикой или отвергает её с изложением логики принятия решения и вновь возвращает откорректированный черновик для дальнейшего рассмотрения. Этот цикл продолжается до тех пор, пока авторы и читатели не придут к единому мнению.

— *Официальное утверждение модели.* Утверждение согласованной модели происходит руководителем рабочей группы в том случае, если у

авторов модели и читателей отсутствуют разногласия по поводу её адекватности. Окончательная модель является согласованным представлением о системе с заданной точки зрения и для заданной цели.

3.2. Синтаксис и семантика моделей SADT/IDEF0

Нотация IDEF0 крайне проста. Она содержит только две сущности — блоки и стрелки.

Функциональные блоки (Activity Box) задают действия. Функциональный блок графически изображается в виде прямоугольника. Он задаёт некоторую конкретную функцию в рамках рассматриваемой системы. По требованиям стандарта название каждого функционального блока должно быть сформулировано в глагольном наклонении (например, «проверить документацию», а не «проверка документации»).

Для выполнения действия могут потребоваться входные данные (сырьё, информация и т.д.). В результате мы получаем что-либо на выходе.

3.2.1. Интерфейсные дуги

Потоки информации обозначаются *интерфейсными дугами (называемые также потоками или стрелками) (Arrow)*. Интерфейсная дуга отображает элемент системы, который обрабатывается функциональным блоком или оказывает иное влияние на функцию, отображённую данным функциональным блоком (см. рис. 3.1).

Графическим отображением интерфейсной дуги является однонаправленная стрелка. Каждая интерфейсная дуга должна иметь своё *уникальное наименование (Arrow Label)*. По требованию стандарта, наименование должно быть оборотом существительного. Началом и концом каждой функциональной дуги могут быть только функциональные блоки, при этом источником может быть только выходная сторона блока, а приёмником — любая из трёх остав-

шихся. Каждый функциональный блок должен иметь, по крайней мере, одну управляющую интерфейсную дугу и одну исходящую.

Типизацию категорий информации можно описать аббревиатурой ICOM:

I (Input), вход — то, что потребляется в ходе выполнения процесса;

C (Control), управление — ограничения и инструкции, влияющие на выполнение процесса;

O (Output), выход — то, что является результатом выполнения процесса;

M (Mechanism), исполняющий механизм — то, что используется для выполнения процесса, но остаётся неизменным.

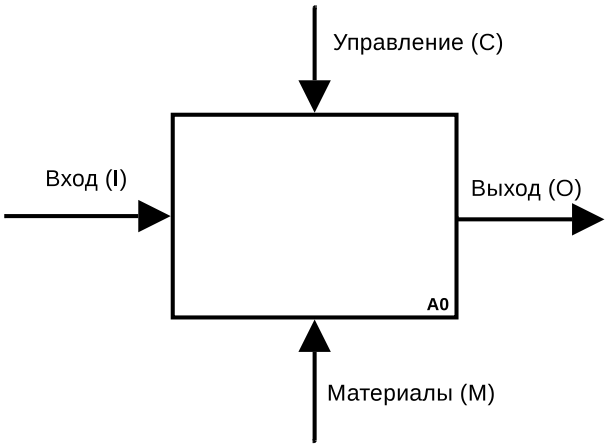


Рис. 3.1. Функциональный блок и интерфейсные дуги

Стрелки входа указывают на сырьё или информацию, потребляемые или преобразуемые функциональным блоком для производства чего-либо на выходе. Поскольку возможно существование блока, ничего не преобразующего и не изменяющего, то наличие входных стрелок не является обязательным. Стрелки входа направлены в левую сторону прямоугольника.

Стрелки управления отвечают за управлением тем, когда и как выполняется функциональный блок. Поскольку управление контролирует поведение функционального блока при создании чего-либо на выходе, то как минимум одна стрелка управления должна присутствовать у каждого блока. Стрелка управления направлена в верхнюю сторону прямоугольника.

Управление остаётся неизменным при работе блока. Если же некая инструкция или правило должно быть изменено блоком, то соответствующую информацию следует рассматривать не как управление, а как входные данные. В случае, когда неясно, относить стрелку к входу или к управлению, следует отнести её к управлению, вплоть до разрешения неясности.

Стрелки выхода указывают на наличие продукции или информации, получаемых в результате работы функционального блока. У каждого блока должен быть хотя бы один выход. Стрелки выхода направлены из правой стороны прямоугольника.

Стрелки механизма исполнения указывают на ресурсы, которые непосредственно выполняют моделируемое действие (например, персонал, техника, оборудование). Они могут отсутствовать, если не являются необходимыми для достижения поставленной цели моделирования. Стрелки механизма исполнения направлены в нижнюю сторону прямоугольника.

3.2.2. Комбинированные стрелки

Выделяют пять основных видов комбинированных стрелок: выход–вход, выход–управление, выход–механизм исполнения, выход–обратная связь на управление, выход–обратная связь на вход.

Стрелка *выход–вход* применяется, когда один из блоков должен полностью завершить работу перед началом работы другого блока (рис. 3.2).

Рис. 3.2. Комбинированная стрелка выход–вход

Стрелка *выход–управление* показывает, что один блок управляет работой другого (рис. 3.3).

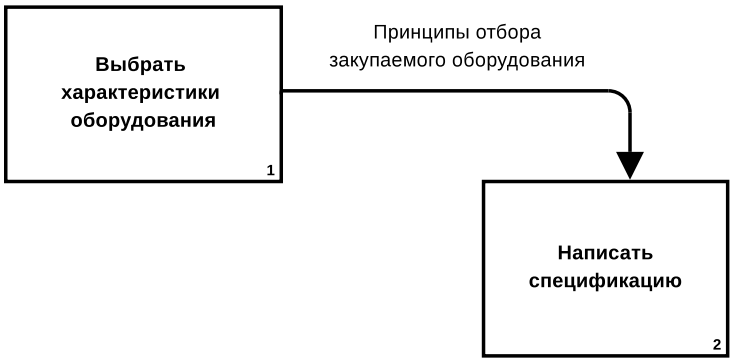


Рис. 3.3. Комбинированная стрелка выход–управление

Стрелки *выход–механизм исполнения* показывают, что выход одного функционального блока применяется в качестве инструментария для работы второго (рис. 3.4).

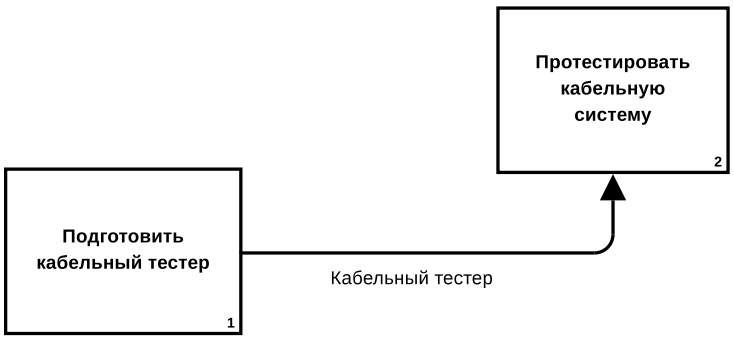


Рис. 3.4. Комбинированная стрелка выход–механизм исполнения

Стрелка *выход–обратная связь на управление* применяется в случае, когда зависимый блок корректирует исполнение управляющего блока (рис. 3.5).

Стрелка *выход–обратная связь на вход* обычно применяется для описания циклов повторной обработки чего-либо (рис. 3.6). Стрелка изображается под блоком. Кроме того, возможно применение данной связи при повторном использовании бракованной продукции.

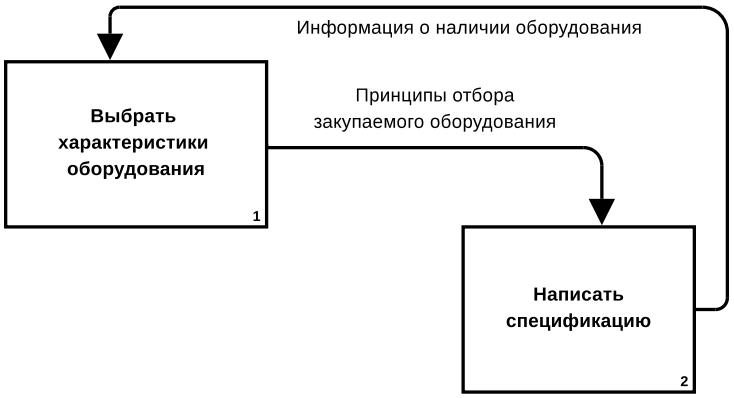


Рис. 3.5. Комбинированная стрелка выход–обратная связь на управление

Рис. 3.6. Комбинированная стрелка выход–обратная связь на вход

3.2.3. Разъединение и соединение стрелок

Выход функционального блока может быть использован в нескольких блоках. В IDEF0 предусматривается соединение и разъединение стрелок. Разъединённые или объединённые стрелки могут иметь наименования, отличающиеся от наименования исходной стрелки. Совокупность исходной и разъединённых или объединённых стрелок называется *связанными стрелками*. Эта техника применяется для того, чтобы отразить использование только части сырья или информации, обозначаемых исходной стрелкой (рис. 3.7).

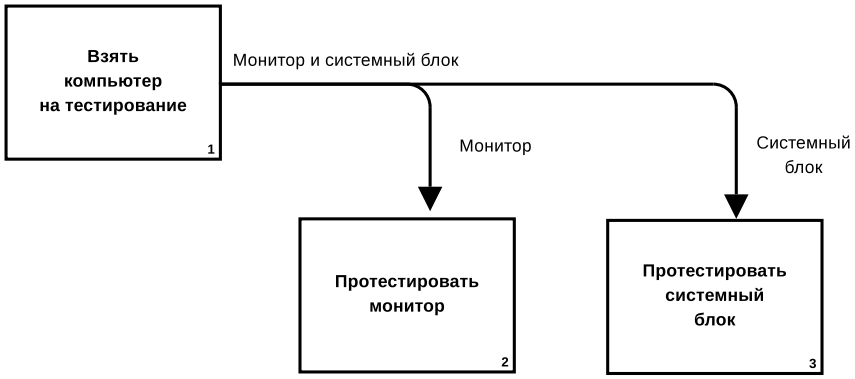


Рис. 3.7. Разъединение и переименование стрелок

3.2.4. Туннели

По методике все элементы, присутствующие на вышележащих диаграммах, должны присутствовать и на нижележащих. Но это может загромождать разработку излишними подробностями. Для управления уровнем детализации используются *туннели*. Если одна из стрелок отсутствует на родительской диаграмме (обычно в связи с несущественностью на определённом уровне абстракции) и не связана с другими стрелками родительской диаграммы, то точка входа или выхода этой стрелки обозначается туннелем. Графически это обозначается обрамлением соответствующего конца стрелки круглыми скобками. Стрелка, выходящая из туннеля, называется *стрелкой импорта ресурсов* (рис. 3.8).

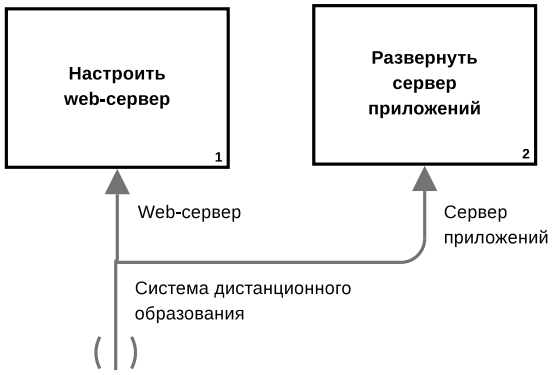


Рис. 3.8. Стрелка, выходящая из туннеля

Стрелка, входящая в туннель, называется *стрелкой подразумевания ресурса* (рис. 3.9).

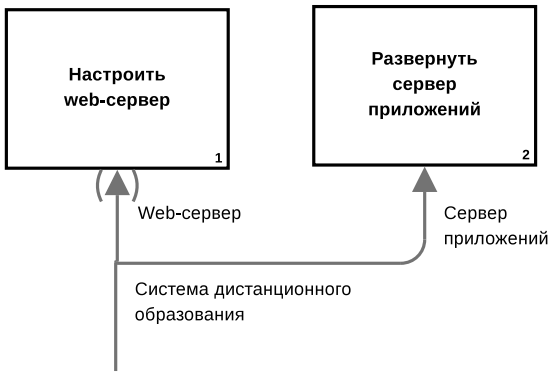


Рис. 3.9. Стрелка, входящая в туннель

3.3. Пример. Метамодель IDEF0

В качестве примера опишем метамодель IDEF0, то есть обозначим элементы диаграммы наиболее общими понятиями и проведём декомпозицию [11]. Для уменьшения объёма составим только контекстную диаграмму и её декомпозицию.

Вначале рисуется контекстная диаграмма, состоящая, как правило, из одного блока (рис. 3.10). Здесь изображены все элементы в общем виде.

Входной поток разбит на группы:

— материальные потоки;
— информационные потоки;
— энергетический поток;
— финансовый поток.

Аналогично разбит и выходной поток:

— материальные потоки;
— информационные потоки;
— финансовый поток.

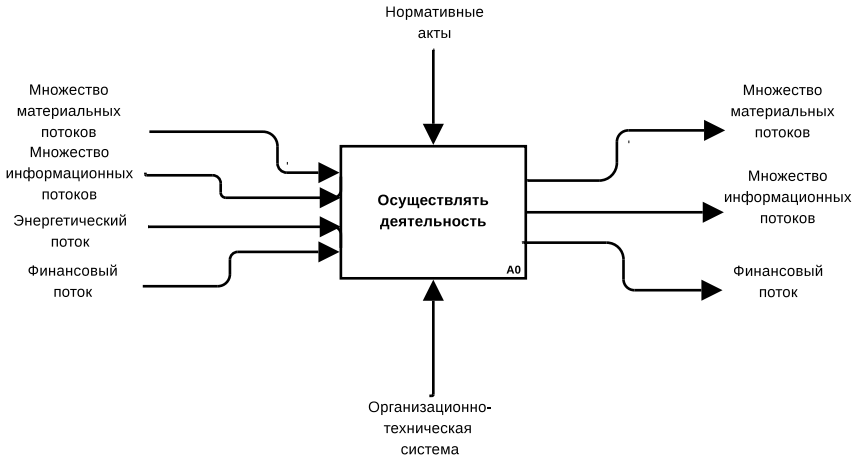


Рис. 3.10. Контекстная диаграмма

Далее происходит первая декомпозиция — декомпозиция контекстной диаграммы (рис. 3.11). На ней представлены основные действия, которые должны быть выполнены для реализации модели.

Далее производится декомпозиция каждой диаграммы в отдельности. Предоставляем читателю возможность это сделать самостоятельно.

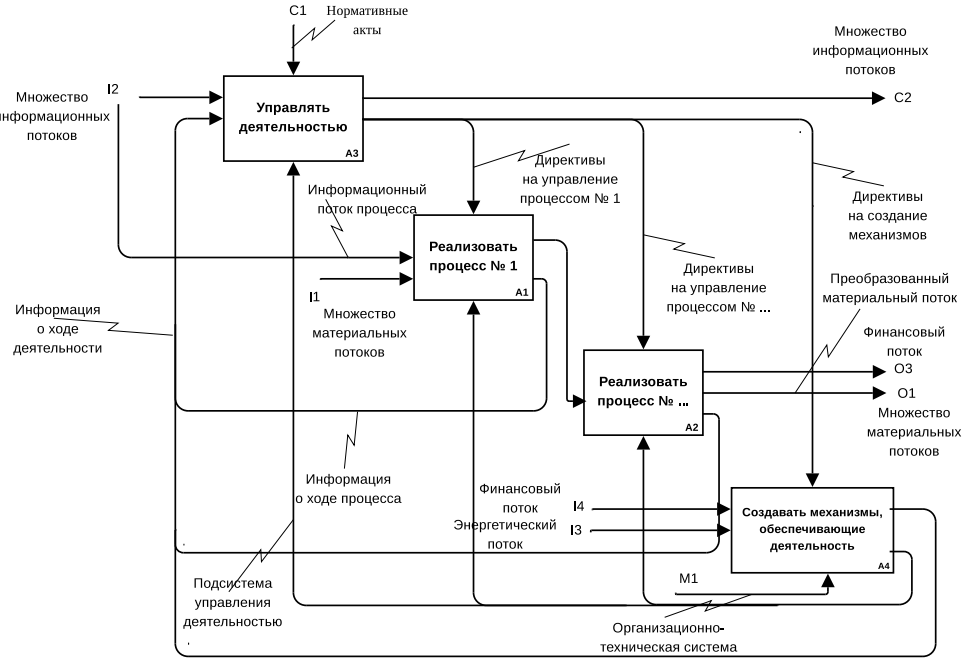


Рис. 3.11. Декомпозиция контекстной диаграммы

Глава 4. Методики IDEF1 и IDEF1X. Информационная модель и модель данных

4.1. Область применения IDEF1

С абстрактной точки зрения при любом роде деятельности производится обработка информации. Движение информации и её изменение называют *информационными потоками*. Любому бизнес-процессу должен соответствовать определённый информационный поток. Управление информационными потоками называется *информационным менеджментом*.

Стандарт IDEF1 был разработан как инструмент для анализа и изучения взаимосвязей между информационными потоками. Целью является структуризация существующей информации. Применение этой методологии позволяет решить следующие задачи:

— выяснить структуру и содержание существующих потоков информации;
— определить, какие проблемы вызваны недостатком управления соответствующей информацией;
— выявить информационные потоки, требующие дополнительного управления для эффективной реализации модели.

Модель IDEF1 включает в рассмотрение не только автоматизированные компоненты, базы данных и соответствующую им информацию, но также и реальные объекты, такие как сотрудники, помещения и т. д. В отличие от методов разработки структур баз данных (например, IDEF1X), IDEF1 является аналитическим методом и используется для выполнения следующих действий:

— определение самой информации и структуры её потоков;
— определение существующих правил и законов, по которым осуществ-

ляется движение информационных потоков, а также принципов управления ими;

— выяснение взаимосвязей между существующими информационными потоками;
— выявление проблем, возникающих вследствие недостатка качественного информационного менеджмента.

Результаты анализа информационных потоков могут быть использованы для стратегического и тактического планирования деятельности и улучшения информационного менеджмента.

Обычно методологию IDEF1 используют при исследовании движения потоков информации и принципов управления ими на начальном этапе процесса проектирования корпоративной информационно-аналитической системы.

Методология IDEF1 позволяет на основе простых графических изображений моделировать информационные взаимосвязи и различия между:

— реальными объектами;
— физическими и абстрактными зависимостями, существующими среди реальных объектов;
— информацией, относящейся к реальным объектам;
— структурой данных, используемой для приобретения, накопления, применения и управления информацией.

4.2. Концепции IDEF1

При построении информационной модели изучаются две предметные области:

1) совокупность физических и интеллектуальных объектов, таких как люди, места, вещи, идеи и т.д., а также все свойства этих объектов и зависимости между ними;
2) информационная область, включающая в себя существующие информационные отображения объектов первой области и их свойств.

Таким образом, IDEF1 есть инструмент для исследования соответствия вышеуказанных областей и установления строгих правил и механизмов изменения объектов информационной области при изменении соответствующих им объектов реального мира.

4.3. Область применения IDEF1X

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис для построения концептуальной схемы. *Концептуальной схемой* называется универсальное представление структуры данных, независимое от конечной реализации базы данных и аппаратной платформы. Будучи статическим методом разработки, IDEF1X изначально не предназначен для динамического анализа по принципу «как есть», тем не менее, он иногда применяется в этом качестве как альтернатива методу IDEF1.

Методика IDEF1X разработана для построения реляционных информационных систем, поскольку:

— требует от проектировщика определить ключевые атрибуты, чтобы отличить одну сущность от другой;

— в тех случаях, когда более чем один атрибут является однозначно идентифицирующим сущность, проектировщик должен определить один из этих атрибутов первичным ключом, а все остальные вторичными.

При совпадении терминологий IDEF1X и IDEF1 их теоретические концепции имеют отличия. Сущность в IDEF1X описывает собой совокупность или набор экземпляров, похожих по свойствам, но однозначно отличаемых друг от друга по одному или нескольким признакам. Каждый экземпляр является реализацией сущности. Таким образом, сущность в IDEF1X описывает конкретный набор экземпляров реального мира, в отличие от сущности в IDEF1, которая представляет собой абстрактный набор информационных отображений реального мира.

4.4. Синтаксис и семантика IDEF1

Методология IDEF1 разделяет элементы структуры информационной области, их свойства и взаимосвязи на классы. Центральным понятием методологии IDEF1 является понятие сущности. *Класс сущностей* представляет собой совокупность информации, накопленной и хранящейся в рамках предприятия и соответствующей определённому объекту или группе объектов реального мира. Основными концептуальными свойствами сущностей в IDEF1 являются:

— устойчивость — информация, имеющая отношение к той или иной сущности, постоянно накапливается;
— уникальность — любая сущность может быть однозначно идентифицирована из другой сущности.

Каждая сущность имеет своё *имя* и *атрибуты* (рис. 4.1).

Рис. 4.1. Сущность IDEF1

Атрибуты представляют собой характерные свойства и признаки объектов реального мира, относящихся к определённой сущности. Класс атрибутов представляет собой набор пар, состоящих из имени атрибута и его значения для определённый сущности. Атрибуты, по которым можно однозначно отличить одну сущность от другой, называются *ключевыми атрибутами*. Каждая сущность может характеризоваться несколькими ключевыми атрибутами. Класс взаимосвязей в IDEF1 представляет собой совокупность взаимосвязей

между сущностями (рис. 4.2). Взаимосвязь между двумя отдельными сущностями считается существующей в том случае, когда класс атрибутов одной сущности содержит ключевые атрибуты другой сущности. Каждый из вышеописанных классов, согласно методологии IDEF1, имеет своё условное графическое отображение.

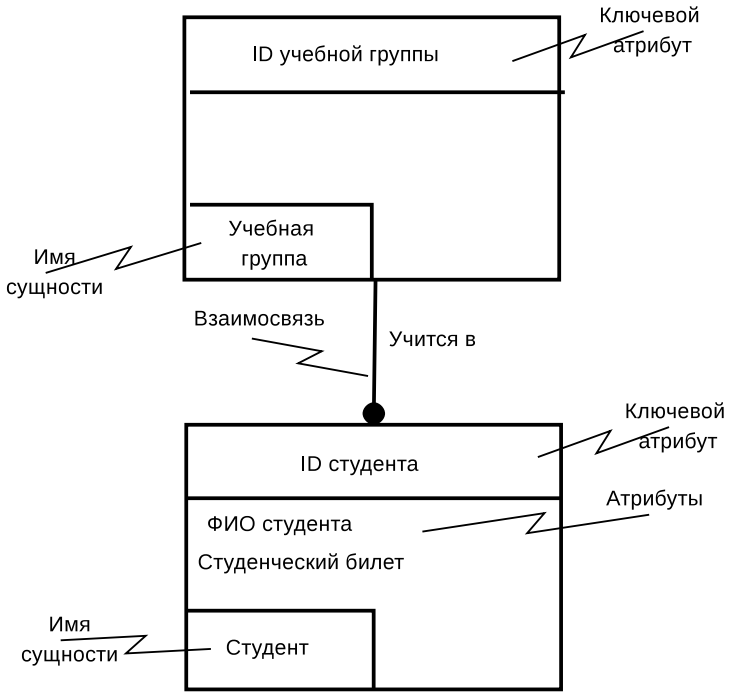


Рис. 4.2. Взаимосвязь сущностей IDEF1

Имя взаимосвязи всегда выражается в глагольной форме. Если же между двумя или несколькими объектами реального мира не существует установленной зависимости, то с точки зрения IDEF1 между соответствующими им сущностями взаимосвязь также отсутствует.

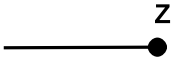
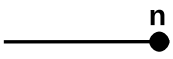
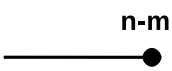
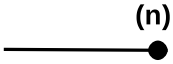
4.5. Синтаксис и семантика IDEF1X

Связи в IDEF1X представляют собой ссылки, соединения и ассоциации между сущностями. Связи — это суть глаголы, которые показывают, как соотносятся сущности между собой. Связи отображаются в виде линии между

двумя сущностями с точкой на одном конце и глагольной фразой, отображаемой над линией. В зависимости от типа отношения связи могут иметь разную мощность (табл. 4.1).

Таблица 4.1

Мощность связей

Связь	**Мощность связи**
	ноль, один или больше
	один или больше
	ноль или один
	точно «n»
	от «n» до «m»
	мощность связи описана в тексте примечания «n»

Сущность описывается в диаграмме IDEF1X графическим объектом в виде прямоугольника. Каждый прямоугольник, отображающий собой сущность, разделяется горизонтальной линией на часть, в которой расположены ключевые поля, и часть, где расположены неключевые поля. Верхняя часть называется *ключевой областью*, а нижняя часть — *областью данных*.

Ключевая область содержит *первичный ключ для сущности* — набор атрибутов, выбранных для идентификации уникальных экземпляров сущности. Атрибуты первичного ключа располагаются над линией в ключевой области. Как следует из названия, неключевой атрибут — это атрибут, который не был выбран ключевым. Неключевые атрибуты располагаются под чертой, в области данных (рис. 4.3). Если сущности в IDEF1X диаграмме связаны, то связь передаёт ключ (или набор ключевых атрибутов) дочерней сущности. Эти атрибуты называются *внешними ключами*. Внешние ключи определяются как

атрибуты первичных ключей родительского объекта, переданные дочернему объекту через их связь. Передаваемые атрибуты называются *мигрирующими*.

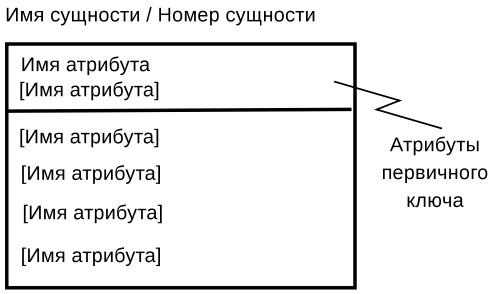


Рис. 4.3. Структура сущности

При разработке модели зачастую приходится сталкиваться с сущностями, уникальность которых зависит от значений атрибута внешнего ключа. Для этих сущностей (для уникального определения каждой сущности) внешний ключ должен быть частью первичного ключа дочернего объекта.

Дочерняя сущность, уникальность которой зависит от атрибута внешнего ключа, называется *зависимой сущностью*. В обозначениях IDEF1X зависимые сущности представлены в виде закруглённых прямоугольников (рис. 4.4).

Рис. 4.4. Зависимая от идентификатора сущность

Зависимые сущности далее классифицируются на сущности, которые не могут существовать без родительской сущности, и сущности, которые не могут быть идентифицированы без использования ключа родителя (сущности, зависящие от идентификации). Сущности, независящие при идентификации от других объектов в модели, называются независимыми сущностями. В IDEF1X независимые сущности представлены в виде прямоугольников (рис. 4.5).

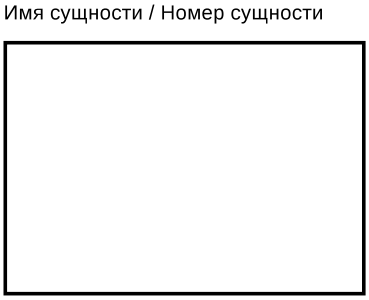


Рис. 4.5. Независимая от идентификатора сущность

Концепция зависимых и независимых сущностей усиливается типом взаимосвязей между двумя сущностями. При передаче внешнего ключа в дочернюю сущность создаётся идентифицирующая связь между родительской и дочерней сущность. Идентифицирующие взаимосвязи обозначаются сплошной линией между сущностями (табл. 4.2).

Таблица 4.2

Типы связей

Связь	Тип связи
—●—	идентифицирующая связь
– – –●	неидентифицирующая связь

Неидентифицирующие связи, являющиеся уникальными для IDEF1X, также связывают родительскую сущность с дочерней. Неидентифицирующие связи используются для отображения другого типа передачи атрибутов внешних ключей — передачи в область данных дочерней сущности (под линией).

Неидентифицирующие связи отображаются пунктирной линией между объектами. Так как переданные ключи в неидентифицирующей связи не являются составной частью первичного ключа дочерней сущности, то этот вид связи не проявляется ни в одной идентифицирующей зависимости.

Основным преимуществом IDEF1X, по сравнению с другими многочисленными методами разработки реляционных баз данных, такими как ER и

ENALIM, является жёсткая и строгая стандартизация моделирования. Установленные стандарты позволяют избежать различной трактовки построенной модели, что является значительным недостатком ER.

Глава 5. Методика IDEF3. Модель процессов

5.1. Назначение методологии

IDEF3 предлагает структурный метод описания процессов. Модель описывается как упорядоченная последовательность событий. Методика IDEF3 хорошо приспособлена для сбора данных.

Данная методика не имеет жёстких синтаксических и семантических ограничений. Очень часто IDEF3 используют как метод, дополняющий IDEF0. Каждый функциональный блок IDEF0 может быть представлен в виде отдельного процесса IDEF3.

5.2. Синтаксис и семантика

Основой методологии является *сценарий (Scenario)* бизнес-процесса, осуществляющий описание последовательности изменений свойств объекта в рамках рассматриваемого процесса. Исполнение каждого сценария сопровождается соответствующим документооборотом, который состоит из двух основных потоков — документов, определяющих структуру и последовательность процесса, и документов, отображающих ход его выполнения.

5.2.1. Диаграммы

В IDEF3 используется два типа диаграмм, представляющие описание одного и того же сценария в разных ракурсах.

1) С помощью *диаграмм описания последовательности этапов процесса (Process Flow Description Diagrams, PFDD)* документируется последовательность и описание стадий обработки в рамках исследуемого бизнес-процесса. Описание производится с точки зрения стороннего на-

блюдателя. Ключевыми элементами являются понятия, процесс, логика процесса.

2) *Диаграммы перехода состояния объекта (Object State Transition Network, OSTN)* используются для иллюстрации трансформаций, которые происходят на каждой стадии бизнес-процесса. При этом описание производится с точки зрения самого объекта.

5.2.2. Единица работы

Действие в IDEF3 называется *единицей работы (Unit of Work, UOW)* и обозначается прямоугольником. Действия именуются глаголами или отглагольными существительными. Каждому действию назначается уникальный номер (рис. 5.1).

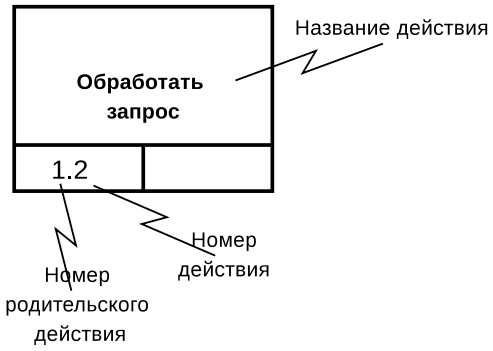


Рис. 5.1. Единица работы

5.2.3. Связи

С помощью связей выделяются существенные взаимоотношения между действиями, задавая их последовательность. Все связи однонаправленные. Обычно стрелки рисуют слева направо, выходящими из правой и входящими в левую сторону блоков, либо сверху вниз (что является лишь соглашением, а не нормируется).

Выделяют три вида связей (табл. 5.1):

— связь *Временно́е предшествование* показывает, что исходное действие должно завершиться прежде, чем начнётся выполнение конечного действия, а также, что исходное действие может инициировать выполнение конечного действия;
— связь *Объектный поток* применяется в случае, когда объект, являющийся результатом выполнения исходного действия, необходим для выполнения конечного действия;
— связь *Нечёткое отношение* используется, если невозможно применить предыдущие типы связей.

При помощи связи типа *Нечёткое отношение* можно, например, описывать параллельные процессы. Корректная интерпретация нечётких связей должна быть документирована в модели.

Таблица 5.1

Типы связей

Графический символ	**Название элемента**	**Назначение**
⟶	Временно́е предшествование (Temporal Precedence)	Исходное действие должно завершиться прежде, чем конечное действие сможет начаться.
⟶▸	Объектный поток (Object Flow)	Выход исходного действия является входом конечного действия. Соответственно исходное действие должно завершиться до начала конечного.
– – –➤	Нечёткое отношение (Relationship)	Вид взаимодействия задаётся отдельно для каждого случая использования.

5.2.4. Соединения

Одно действие может порождать несколько. Или для выполнения действия требуется завершение нескольких действий. Для описания ветвлений процессов используют соединения (табл. 5.2).

Таблица 5.2

Соединения IDEF3

Графический символ	**Название элемента**	**Смысл в случае слияния стрелок** (Fan-in Junction)	**Смысл в случае разветвления стрелок** (Fan-out Junction)
	Асинхронное «И» (Asynchronous AND)	Все предшествующие процессы должны быть завершены.	Все следующие процессы должны быть запущены.
	Синхронное «И» (Synchronous AND)	Все предшествующие процессы завершаются одновременно.	Все следующие процессы запускаются одновременно.
	Асинхронное «ИЛИ» (Asynchronous OR)	Один или несколько предшествующих процессов должны быть завершены.	Один или несколько следующих процессов должны быть запущены.
	Синхронное «ИЛИ» (Synchronous OR)	Один или несколько предшествующих процессов завершаются одновременно.	Один или несколько следующих процессов запускаются одновременно.
	Исключающее «ИЛИ» (XOR — Exclusive OR)	Только один предшествующий процесс завершён.	Только один следующий процесс запускается.

Соединения разбиваются по следующим дихотомиям.

1) Разворачивающие и сворачивающие соединения (ветвление соединений):

— разворачивающие соединения используют для разъединения потоков, так что завершение одного действия вызывает начало нескольких других;

— сворачивающие соединения объединяют потоки, то есть завершение одного или нескольких действий вызывает начало выполнения другого.

2) Синхронные и асинхронные соединения. Иногда необходимо начинать или заканчивать параллельные действия одновременно, что изображается с помощью синхронных соединений. В противном случае соединение является асинхронным.

Все соединения на диаграмме должны быть парными. Однако при этом типы соединений не обязаны совпадать. На диаграмме соединения обычно обозначаются буквой «J» и цифрой (рис. 5.2).

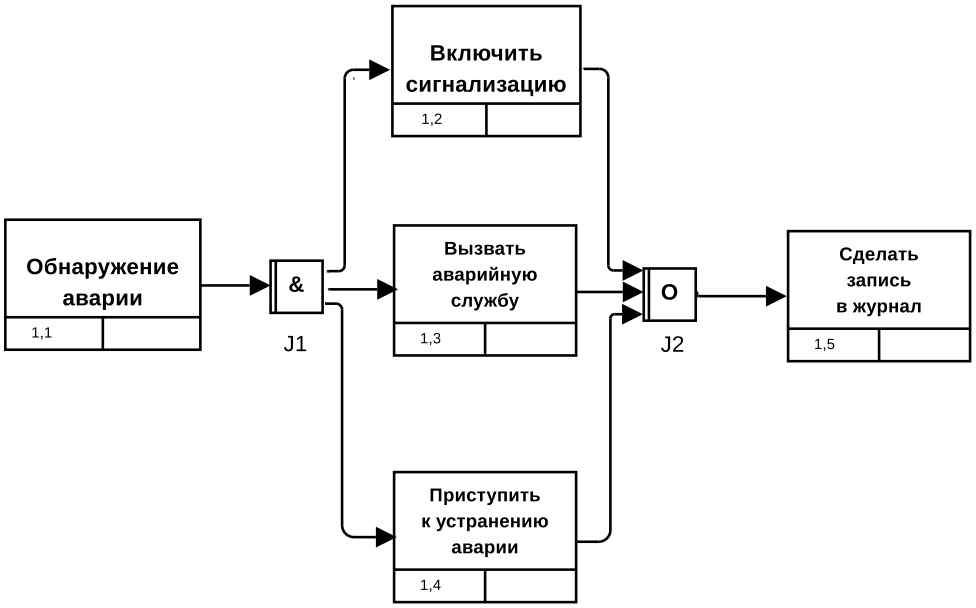


Рис. 5.2. Пример комбинации двух типов соединений

Глава 6. Другие методики IDEF

6.1. Методика IDEF2. Имитационная модель

IDEF2 — методология имитационного моделирования развития систем. В связи с весьма серьёзными сложностями анализа динамических систем от этого стандарта практически отказались, и его развитие приостановилось на самом начальном этапе. Однако в настоящее время применяются алгоритмы и их компьютерные реализации, позволяющие превращать набор статических диаграмм IDEF0 в динамические модели, построенные на базе *«раскрашенных сетей Петри» (Color Petri Nets, CPN)*.

В IDEF2 модель разбивается на четыре подмодели:

— *подмодель возможностей*, которая описывает агентов;

— *подмодель потока сущностей*, которая описывает трансформацию сущностей;

— *подмодель распределения ресурсов*, которая описывает распределение агентов для проведения трансформаций;

— *подмодель управления системой*, которая описывает внешние воздействия.

Преимуществом методики является то, что набор диаграмм может быть непосредственно переведён в имитационную модель.

Наверное, основной проблемой на пути развития и внедрения этой методологии является то, что бизнес-аналитики обычно плохо знают имитационное моделирование (и наоборот).

6.2. Методика IDEF4. Объектно-ориентированные методы проектирования

Методология IDEF4 вводит объектно-ориентированный подход в набор стандартов IDEF. Будучи одной из прародительниц методики UML, она была отодвинута впоследствии на периферию и сейчас практически не применяется.

6.2.1. Концепции IDEF4

Следуя общей методологии, IDEF4 предлагает разбивать модель на набор диаграмм, а не пытаться втиснуть всё в одну диаграмму. IDEF4 предлагает целую методологию объектно-ориентированного дизайна, а не просто графический синтаксис.

Модель IDEF4 разбивается на две подмодели: *подмодель классов* и *подмодель методов* (рис. 6.1).

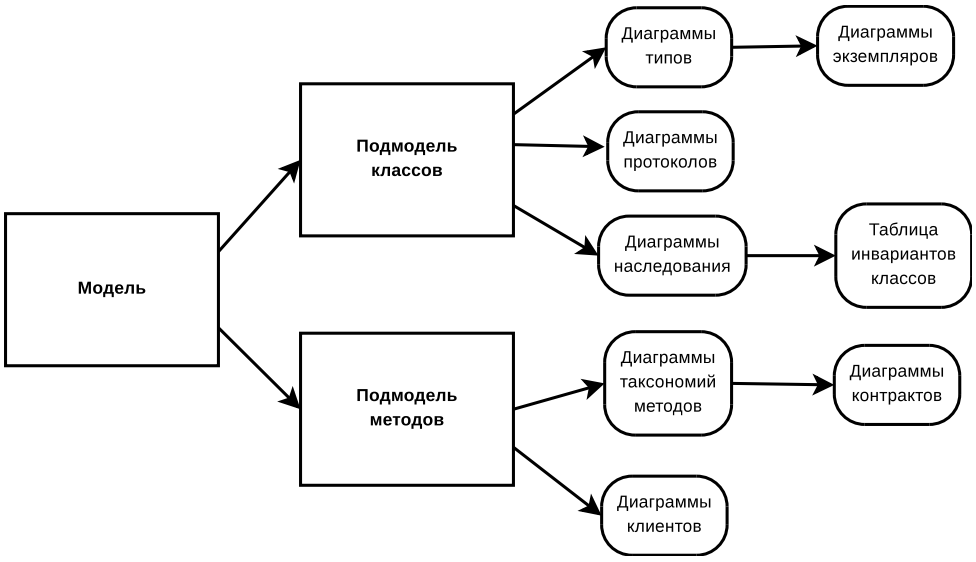


Рис. 6.1. Структура модели IDEF4

Подмодель класса декомпозируется, в свою очередь, на следующие диаграммы:

— диаграмму наследования;
— диаграмму типов;
— диаграмму протоколов;
— диаграмму экземпляров.

Подмодель методов декомпозируется на следующие диаграммы:

— диаграмму таксономий методов;
— диаграмму клиентов.

6.2.2. Синтаксис и семантика моделей IDEF4

6.2.2.1. Подмодель классов IDEF4

Подмодель классов описывает структуру классов и их наследование.

Диаграммы наследования описывают порядок наследования классов. Например, класс *Filled-Rectangle* наследует структуру и поведение непосредственно от классов *Rectangle* и *Filled-Object*, которые, в свою очередь, наследуют классу *Object* (рис. 6.2). Диаграммы наследования уточняются с помощью таблиц инвариантов классов.

Диаграммы экземпляров связаны с *диаграммами типов* и используются с целью их уточнения (например, если между ними существуют сложные взаимосвязи) (рис. 6.3).

Диаграммы протоколов описывают типы аргументов классов при вызове методов. На рис. 6.4 дана диаграмма протоколов для объекта *Fill-Closed-Object*. Он воспринимает экземпляр класса *Polygon* в качестве первичного аргумента и экземпляр класса *Color* — в качестве второго, и возвращает экземпляр себя классу *Polygon*.

6.2.2.2. Подмодель методов IDEF4

Диаграмма таксономий методов классифицирует методы по подобности поведения. На рис. 6.5 метод *Print* выражает контракт, что состояние метода

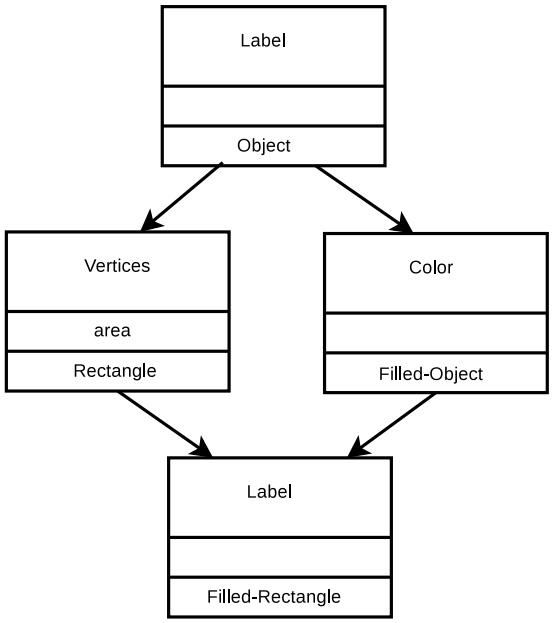


Рис. 6.2. Пример диаграммы наследования IDEF4

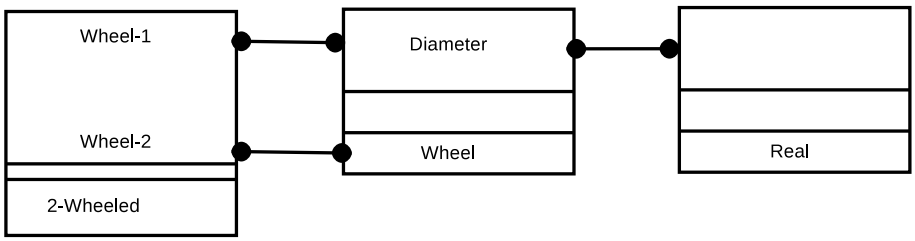


Рис. 6.3. Пример диаграммы типов IDEF4

должно быть печатаемо, и быть либо печатаемым текстом, либо печатаемой графикой.

Диаграммы клиентов связывают вызывающие и вызываемые процедуры. Сдвоенные стрелки направлены от вызываемой к вызывающей процедуре. На рис. 6.6 процедура *Redisplay* вызывает процедуру *Erase* объекта *Erasable-Object* и процедуру *Draw* объекта *Drawable-Object*.

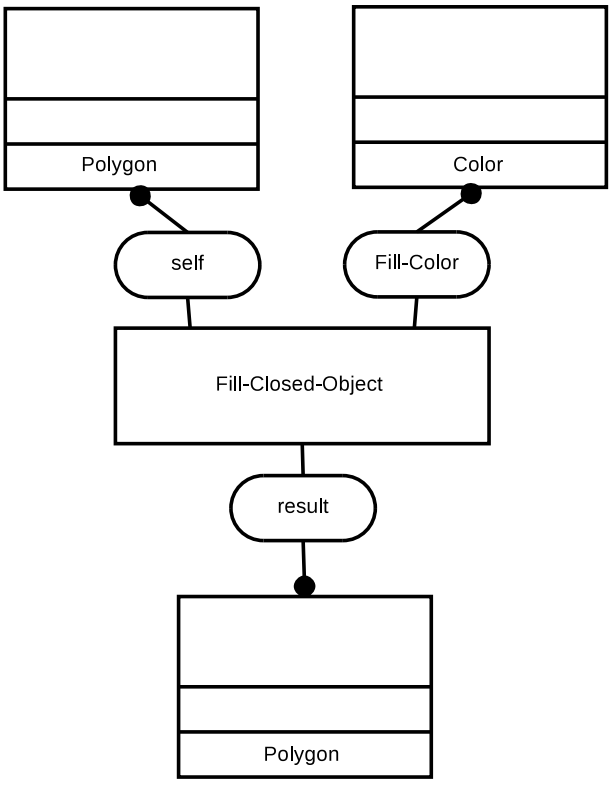


Рис. 6.4. Пример диаграммы протоколов IDEF4

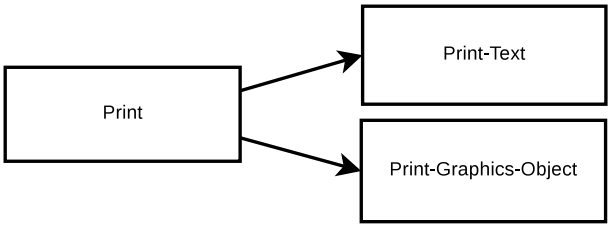


Рис. 6.5. Пример диаграммы таксономий методов IDEF4

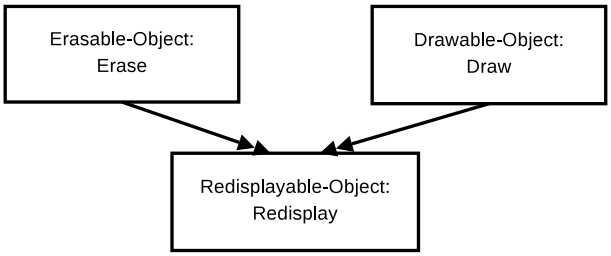


Рис. 6.6. Пример диаграммы клиент IDEF4

Глава 7. Методика DFD. Диаграммы потоков данных

7.1. Методология DFD

Методика IDEF0 может дополняться не только диаграммами потока работ (IDEF3), но и *диаграммами потоков данных (Data Flow Diagramming, DFD)*.

Нотация диаграмм потоков данных позволяет отображать на диаграмме как шаги бизнес-процесса, так и поток документов и управления (в основном управления, поскольку на верхнем уровне описания процессных областей значение имеет передача управления). Также на диаграмме можно отображать средства автоматизации шагов бизнес-процессов. Обычно используется для отображения третьего и ниже уровня декомпозиции бизнес-процессов (первый уровень — перечень бизнес-процессов (IDEF0), а второй — функции, выполняемые в рамках бизнес-процессов (IDEF3)).

Области применения диаграмм потоков данных:

— моделирование функциональных требований к проектируемой системе;
— моделирование существующего процесса движения информации;
— описание документооборота, обработки информации;
— дополнение к модели IDEF0 для более наглядного отображения текущих операций документооборота;
— проведение анализа и определения основных направлений реинжиниринга информационной системы.

Диаграммы DFD могут дополнить то, что уже отражено в модели IDEF0, поскольку они описывают потоки данных, позволяя проследить, каким образом происходит обмен информацией как внутри системы между бизнес-функциями, так и системы в целом с внешней информационной средой.

Методика DFD удобна для описания не только бизнес-процессов (как дополнение к IDEF0), но и программных систем:

— DFD-диаграммы создавались как средство проектирования программных систем (в то время как IDEF0 — средство проектирования систем вообще), поэтому DFD имеют более богатый набор элементов, адекватно отражающих их специфику (например, хранилища данных являются прообразами файлов или баз данных);

— наличие мини-спецификаций DFD-процессов нижнего уровня позволяет преодолеть логическую незавершённость IDEF0 (моделирование обрывается на некотором достаточно низком уровне, когда дальнейшая детализация модели становится бессмысленной) и построить полную функциональную спецификацию разрабатываемой системы.

С помощью DFD-диаграмм требования к проектируемой информационной системе разбиваются на функциональные компоненты (процессы) и представляются в виде сети, связанной потоками данных. Главная цель декомпозиции DFD-функций — продемонстрировать, как каждый процесс преобразует свои входные данные в выходные, а также выявить отношения между этими процессами.

На схемах бизнес-процесса отображаются:

— функции процесса;

— входящая и исходящая информация при описании документов;

— внешние бизнес-процессы, описанные на других диаграммах;

— точки разрыва при переходе процесса на другие страницы.

При моделировании DFD система рассматривается как сеть связанных между собой функций, т. е. как совокупность сущностей. Методология основана на идее нисходящей иерархической организации. Целью является преобразование общих, неясных знаний о требованиях к системе в точные определения. Внимание фокусируется на потоках данных. Методология поддерживается традиционными нисходящими методами проектирования.

7.1.1. Варианты методологии DFD

Существует два основных варианта методологии DFD: *методология Гейна–Сарсона (Gane–Sarson)* и *методология Йордана–Де Марко (Yourdon–DeMarko)*.

Главной отличительной чертой методологии Гейна-Сарсона является наличие этапа моделирования данных, определяющего содержимое хранилищ данных (БД и файлов) в DFD. Этот этап включает построение списка элементов данных, располагающихся в каждом хранилище данных; анализ отношений между данными и построение соответствующей диаграммы связей между элементами данных; представление всей информации по модели в виде связанных нормализованных таблиц.

Кроме того, эти методологии отличаются нотацией.

Обе методологии основаны на простой концепции нисходящего поэтапного разбиения функций системы на подфункции:

1) формирование контекстной диаграммы верхнего уровня, идентифицирующей границы системы и определяющей интерфейсы между системой и окружением;
2) формирование списка внешних событий, на которые система должна реагировать (после интервьюирования эксперта предметной области), для каждого из которых строится *пустой процесс (Bubble)* в предположении, что его функция обеспечивает требуемую реакцию на это событие (в большинстве случаев включает генерацию выходных потоков и событий);
3) проведение детализации для каждого из пустых процессов.

Помимо нотаций Йордона–Де Марко и Гейна–Сарсона для элементов DFD-диаграм могут использоваться и другие условные обозначения (OMT, SSADM и т. д.). Все они обладают практически одинаковой функциональностью и различаются лишь в деталях.

Методология DFD позволяет уже на стадии функционального моделиро-

вания определить базовые требования к данным. В этом случае совместно используются методологии DFD и IDEF1X.

Диаграммы DFD могут быть построены с использованием традиционного структурного анализа, подобно тому, как строятся диаграммы IDEF0:

1) строится физическая модель, отображающая текущее состояние дел;
2) полученная модель преобразуется в логическую модель, которая отображает требования к существующей системе;
3) строится модель, отображающая требования к будущей системе;
4) строится физическая модель, на основе которой должна быть построена новая система.

Альтернативным является подход, применяемый при создании программного обеспечения, называемый *событийным разделением (Event Partitioning)*, в котором различные диаграммы DFD выстраивают модель системы.

1) Логическая модель строится как совокупность процессов и документирования того, что эти процессы должны делать.
2) С помощью модели окружения система описывается как взаимодействующий с событиями из внешних сущностей объект. *Модель окружения (Environment Model)* обычно содержит описание цели системы, одну контекстную диаграмму и список событий. Контекстная диаграмма содержит один блок, изображающий систему в целом, внешние сущности, с которыми система взаимодействует, ссылки и некоторые стрелки, импортированные из диаграмм IDEF0 и DFD. Включение внешних ссылок в контекстную диаграмму не отменяет требования методологии чётко определить цель, область и единую точку зрения на моделируемую систему.
3) *Модель поведения (Behavior Model)* показывает, как система обрабатывает события. Эта модель состоит из одной диаграммы, в которой каждый блок изображает каждое событие из модели окружения, могут быть добавлены хранилища для моделирования данных, которые необходимо запоминать между событиями. Потоки добавляются для связи

с другими элементами, и диаграмма проверяется с точки зрения соответствия модели окружения.

7.1.2. Мини-спецификация

Мини-спецификация — это алгоритм описания задач, выполняемых процессами, множество всех мини-спецификаций является полной спецификацией системы. Мини-спецификации содержат номер и/или имя процесса, списки входных и выходных данных и тело (описание) процесса, являющееся спецификацией алгоритма или операции, трансформирующей входные потоки данных в выходные.

Проектные спецификации строятся по DFD и их мини-спецификациям автоматически. Наиболее часто для описания проектных спецификаций используется методика структурных карт Джексона, иллюстрирующая иерархию модулей, связи между ними и некоторую информацию об их исполнении (последовательность вызовов, итерацию). Существует ряд методов автоматического преобразования DFD в структурные карты.

7.2. Синтаксис и семантика моделей DFD

Диаграммы потоков данных применяются для графического представления (Flowchart) движения и обработки информации. Обычно диаграммы этого типа используются для проведения анализа организации информационных потоков и для разработки информационных систем. Каждый блок в DFD может развёртываться в диаграмму нижнего уровня, что позволяет на любом уровне абстрагироваться от деталей.

DFD-диаграммы моделируют функции, которые система должна выполнять, но почти ничего не сообщают об отношениях между данными, а также о поведении системы в зависимости от времени — для этих целей используются диаграммы сущность-связь и диаграммы переходов состояний.

Основные объекты нотации DFD:

— *блоки (Blocks)* или *работы (Activities)* — отображают процессы обработки и изменения информации;
— *стрелки (Arrows)* или *потоки данных (Data Flow)* — отображают информационные потоки;
— *хранилища данных (Data Store)* — отображают данные, к которым осуществляется доступ; эти данные используются, создаются или изменяются работами;
— *внешние ссылки (External References)* или *внешние сущности (External Entity)* — отображают объекты, с которыми происходит взаимодействие.

Работы DFD — графическое изображение операции по обработке или преобразованию информации (данных). Входная информация преобразуется в выходную. Работы именуются глаголами в неопределённой форме с последующим дополнением.

По нотации Гейна–Сарсона DFD-блок изображается прямоугольником со скруглёнными углами (рис. 7.1). Каждый блок должен иметь уникальный номер для ссылки на него внутри диаграммы. Номер каждого блока может включать префикс, номер родительского блока (A) и номер объекта, представляющий собой уникальный номер блока на диаграмме. Например, работа может иметь номер A.12.4.

Рис. 7.1. Работа DFD

Во избежание пересечений линий потоков данных один и тот же элемент

может на одной и той же диаграмме отображаться несколько раз; в таком случае два или более прямоугольников, обозначающих один и тот же элемент, могут идентифицироваться линией, перечёркивающей нижний правый угол.

Поток данных — механизм, использующийся для моделирования передачи информации между участниками процесса информационного обмена (функциями, хранилищами данных, внешними ссылками). По нотации Гейна–Сарсона поток данных изображается стрелкой между двумя объектами DFD-диаграммы, предпочтительно горизонтальной и/или вертикальной, причём направление стрелки указывает направление потока. Каждая стрелка должна иметь источник и цель. В отличие от стрелок IDEF0-диаграммы, стрелки DFD могут входить или выходить из любой стороны блока.

Стрелки описывают, как объекты (включая данные) двигаются из одной части системы в другую. Поскольку в DFD каждая сторона блока не имеет чёткого назначения, в отличие от блоков IDEF0-диаграммы, стрелки могут подходить и выходить из любой грани.

В DFD-диаграммах для описания диалогов типа команды-ответа между операциями применяются двунаправленные стрелки между функцией и внешней сущностью и/или между внешними сущностями. Стрелки могут сливаться и разветвляться, что позволяет описать декомпозицию стрелок. Каждый новый сегмент сливающейся или разветвляющейся стрелки может иметь собственное имя.

Иногда информация может двигаться в одном направлении, обрабатываться и возвращаться обратно. Такая ситуация может моделироваться либо двумя различными потоками, либо одним двунаправленным потоком. На поток данных можно ссылаться, указывая процессы, сущности или накопители данных, которые поток соединяет. Каждый поток должен иметь имя, расположенное вдоль или над стрелкой, выбранное таким образом, чтобы в наибольшей степени передавать смысл содержания потока пользователям, которые будут рассматривать диаграмму потоков данных.

Хранилище данных — графическое представление потоков данных, им-

портируемых/экспортируемых из соответствующих баз данных, изображает объекты в покое. Является неким прообразом базы данных информационной системы организации.

Хранилища данных предназначены для изображения неких абстрактных устройств для хранения информации, которую можно в любой момент времени поместить или извлечь из них, безотносительно к их конкретной физической реализации.

Хранилища данных используются:

— в материальных системах (там, где объекты ожидают обработки, например в очереди);
— в системах обработки информации для моделирования механизмов сохранения данных с целью дальнейших операций.

По нотации Гейна–Сарсона хранилище данных обозначается двумя горизонтальными линиями, замкнутыми с одного края (рис. 7.2). Каждое хранилище данных должно идентифицироваться для ссылки буквой D и произвольным числом в квадрате с левой стороны, например D5.

Рис. 7.2. Хранилище данных DFD

В модели может быть создано множество вхождений хранилищ данных, каждое из которых может иметь одинаковое имя и ссылочный номер. Для того чтобы не усложнять диаграмму потоков данных пересечениями линий, можно изображать дубликаты накопителя данных дополнительными вертикальными линиями с левой стороны квадрата. Хранилища данных не изменяют потоки данных, а служат только для хранения поступающих объектов.

Внешняя сущность — объект диаграммы потоков данных, являющийся источником или приёмником информации извне модели. Внешние сущности изображают входы и/или выходы, т. е. обеспечивают интерфейс с внешними

объектами, находящимися вне моделируемой системы. Внешними ссылками системы обычно являются логические классы предметов или людей, представляющие собой источник или приёмник сообщений, например, заказчики, конструкторы, технологи, производственные службы, кладовщики и т. д.

По нотации Гейна–Сарсона пиктограмма внешней ссылки представляет собой оттенённый прямоугольник, верхняя левая сторона которого имеет двойную толщину и обычно располагается на границах диаграммы (рис. 7.3). Внешняя ссылка может идентифицироваться строчной буквой E в левом верхнем углу и уникальным номером, например E5. Кроме того, внешняя ссылка имеет имя.

Рис. 7.3. Внешняя сущность DFD

Одна и та же внешняя ссылка может быть использована многократно на одной или нескольких диаграммах. Обычно к такому приёму прибегают для того, чтобы не рисовать слишком длинных и запутанных стрелок. Каждая внешняя сущность имеет префикс.

При рассмотрении системы как внешней функции часто указывается, что она находится за пределами границ моделируемой системы. После проведения анализа некоторые внешние ссылки могут быть перемещены внутрь диаграммы потоков данных рассматриваемой системы или, наоборот, какая-то часть функций системы может быть вынесена и рассмотрена как внешняя ссылка. Преобразования потоков данных во внешних сущностях игнорируются.

Межстраничные ссылки (Off-Page Reference) — инструмент нотации DFD, описывающий передачу данных или объектов с одной диаграммы модели на

другую. Стрелка межстраничной ссылки имеет идентифицирующее имя, номер и изображение окружности.

Помимо этого, для каждого информационного потока и хранилища определяются связанные с ними элементы данных. Каждому элементу данных присваивается имя. Также для него может быть указан тип данных и формат. Именно эта информация является исходной на следующем этапе проектирования — построении модели «сущность–связь». При этом, как правило, информационные хранилища преобразуются в сущности. Проектировщику остаётся только решить вопрос с использованием элементов данных, не связанных с хранилищами.

В качестве примера построим схему бизнес-процесса «Выдача диплома» (рис. 7.4).

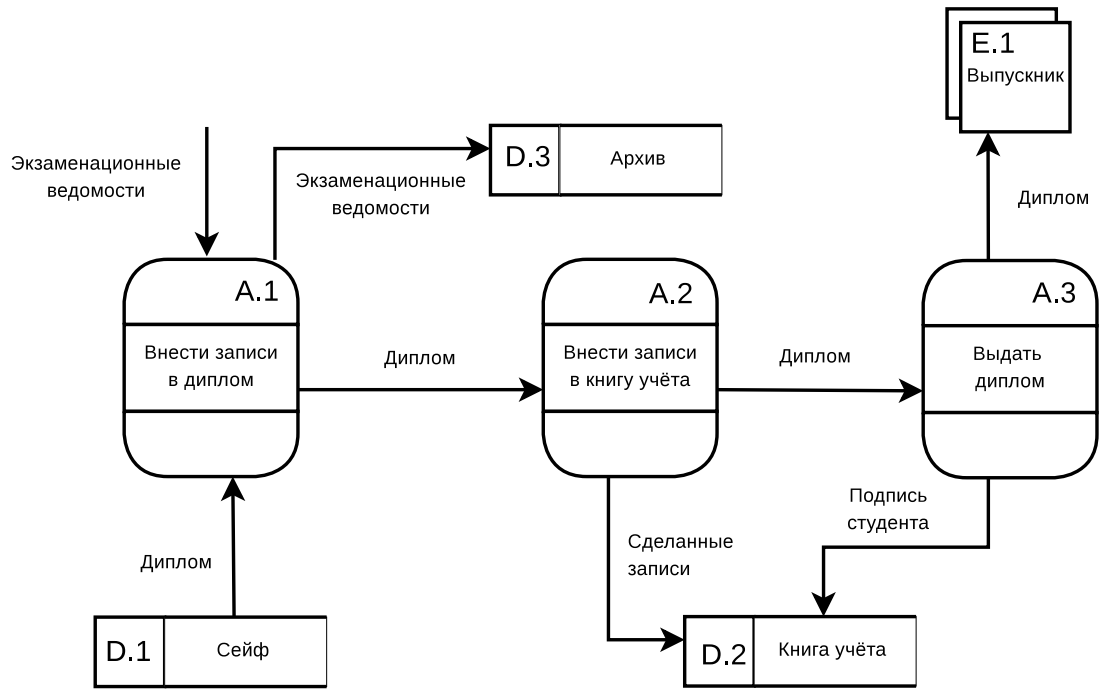


Рис. 7.4. DFD-схема «Выдача диплома»

Глава 8. Универсальный язык UML моделирования сложных систем

8.1. Диаграммы UML

Описание языка UML состоит из двух взаимодействующих частей:

1) *семантики языка UML* — некоторой метамодели, определяющей абстрактный синтаксис и семантику понятий объектного моделирования на языке UML;
2) *нотации языка UML* — графической нотации для визуального представления семантики языка UML.

Семантика определяется для двух видов объектных моделей:

— *структурных моделей (статических моделей)*, которые описывают структуру сущностей или компонентов некоторой системы;
— *моделей поведения (динамических моделей)*, которые описывают поведение или функционирование объектов системы.

Нотация языка UML включает в себя описание отдельных семантических элементов, которые могут применяться при построении диаграмм.

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

— диаграмма вариантов использования (Use Case Diagram);
— диаграмма классов (Class Diagram);
— диаграммы поведения (Behavior Diagrams):
  – диаграмма состояний (Statechart Diagram),
  – диаграмма деятельности (Activity Diagram),

- диаграммы взаимодействия (Interaction Diagrams):
  - диаграмма последовательности (Sequence Diagram),
  - диаграмма кооперации (Collaboration Diagram);
- диаграммы реализации (Implementation Diagrams):
  - диаграмма компонентов (Component Diagram),
  - диаграмма развёртывания (Deployment Diagram).

8.2. Концепция построения диаграмм UML

Процесс построения отдельных типов диаграмм имеет свои особенности, которые тесно связаны с семантикой элементов этих диаграмм. Сам процесс объектно-ориентированного анализа и проектирования в контексте языка UML получил специальное название — *рациональный унифицированный процесс (Rational Unified Process, RUP)*.

Суть концепции RUP заключается в последовательной декомпозиции или разбиении процесса объектно-ориентированного анализа и проектирования на отдельные этапы, на каждом из которых осуществляется разработка соответствующих типов канонических диаграмм модели системы. При этом порядок этапов построения модели следующий:

1) логические представления статической модели структуры системы;
2) логические представления модели поведения;
3) физические представления модели системы.

В результате RUP должны быть построены канонические диаграммы на языке UML, при этом последовательность их разработки в основном совпадает с их последовательной нумерацией.

8.3. Основные элементы UML

Для диаграмм языка UML существуют три типа визуальных обозначений:

1) *связи* — представляются различными линиями на плоскости (табл. 8.1);
2) *графические символы* — изображаются вблизи от тех или иных визуальных элементов диаграмм (табл. 8.2);
3) *текст* — содержится внутри отдельных геометрических фигур, форма которых (прямоугольник, эллипс) соответствует некоторым элементам языка UML (класс, вариант использования) и имеет фиксированную семантику.

Таблица 8.1

Связующие элементы UML

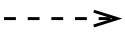
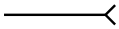
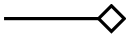
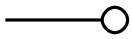
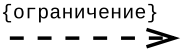
Графический символ	**Название элемента**
	отношение ассоциации
	отношение зависимости
	отношение наследования
	агрегация
	композиция
	приёмник события
	источник события
	замена
	граница
	простой поток управления
	ограничение
	вызов
	рекурсия
	возвратить

Таблица 8.1

Связующие элементы UML (окончание)

Графический символ	**Название элемента**
	послать

Таблица 8.2

Основные графические символы UML

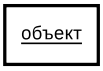
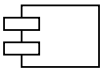
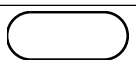
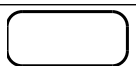
Графический символ	**Название элемента**
	действующее лицо (актёр)
	вариант использования
	примечание
	пакет
	класс
	объект
	компонент
	узел
	кооперация
	деятельность
	состояние
	начальное состояние
	конечное состояние
	контроль

Таблица 8.2

Основные графические символы UML (окончание)

Графический символ	Название элемента
	сущность
	граница
	разветвление
	линия жизни
	разветвление

8.4. Диаграмма вариантов использования

Диаграмма вариантов использования (Use Case Diagram) представляет собой графическое изображение взаимодействия некоторой сущности (действующего лица) и моделируемой системы. Каждый вариант использования охватывает некоторую функцию системы и решает некоторую дискретную задачу, поставленную сущностью перед системой. Список всех вариантов использования фактически определяет функциональные требования к системе. Таким образом, диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе её проектирования и разработки.

При разработке диаграммы вариантов использования должны быть решены следующие задачи:

— должны быть определены общие границы и контекст предметной области моделируемой системы;

— должны быть сформулированы общие требования к функциональному поведению проектируемой системы;

— должна быть разработана исходная концептуальная модель системы.

Основными элементами диаграммы вариантов использования являются *действующее лицо*, *вариант использования*, *интерфейс*, *отношения*, *примечания*.

8.4.1. Действующее лицо

Действующее лицо или актёр (Actor) представляет собой некоторую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует её функциональные возможности для достижения определённых целей. В общем случае актёр находится вне системы, его внутренняя структура никак не определяется.

Действующему лицу ставится в соответствие множество логически связанных ролей, исполняемых при взаимодействии с системой. Под *ролью (Role)* понимается поведение сущности, участвующей в конкретном контексте.

Стандартным графическим обозначением актёра на диаграммах является фигурка «человечка», под которой записывается его имя (рис. 8.1).

Рис. 8.1. Графическое изображение актёра в языке UML

Актёрами являются сущности, имеющие отношение к концептуальной модели соответствующей предметной области: клиент, служащий, сотовый телефон, станция сотовой связи, абонент, автомобиль и пр. В качестве актёров могут выступать также другие системы, подсистемы проектируемой системы или отдельные классы.

Действующие лица взаимодействуют с системой посредством передачи и приёма сообщений от вариантов использования. *Сообщение* представляет собой запрос актёром сервиса от системы и получение этого сервиса. Кроме

того, с актёрами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актёрами.

8.4.2. Вариант использования

Вариант использования (Use Case) представляет собой внешнюю спецификацию последовательности действий, которые система или другая сущность могут выполнять в процессе взаимодействия с действующими лицами.

Варианты использования предназначены для определения функциональных требований к системе. Они служат для описания сервисов, которые система предоставляет актёру. Другими словами, каждый вариант использования определяет конечный набор действий, совершаемый системой при диалоге с актёром, а также описывает реакции сущности системы на получение отдельных сообщений от действующих лиц и восприятие этих сообщений за пределами сущности. При этом реализация самого взаимодействия не определяется.

Вариант использования представляется на диаграмме эллипсом (рис. 8.2), внутри которого или под ним располагается его имя (в форме существительного или глагола).

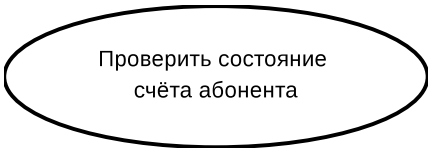


Рис. 8.2. Графическое изображение варианта использования в языке UML

8.4.3. Интерфейс

Интерфейс (Interface) представляет собой именованное множество операций, которые характеризуют поведение отдельного элемента модели. Интерфейсы могут содержать только операции без указания особенностей их реализации.

Интерфейс изображается в виде маленького круга, рядом с которым записывается его имя (рис. 8.3).

Рис. 8.3. Графическое изображение интерфейса в языке UML

Формально интерфейс не только отделяет спецификацию операций системы от их реализации, но и определяет общие границы проектируемой системы. Интерфейсы определяют стыковочные узлы в проектируемой системе и позволяют безболезненно модифицировать уже существующую систему при переходе на новые технологические решения.

8.4.4. Отношения

Отношение (Relationship) представляет собой семантическую связь между отдельными элементами модели. Различные виды отношений в той или иной степени используются при построении всех диаграмм.

В языке UML определены следующие виды отношений между действующими лицами и вариантами использования: *ассоциации*, *расширения*, *включения*, *обобщения*.

Отношение ассоциации (Association Relationship) представляет собой структурное отношение, показывающее, что объекты одного типа некоторым образом связаны с объектами другого типа. В ассоциации могут связываться два или более объектов. Тогда ассоциация называется соответственно *бинарной* или n*-арной*. Графически отношение ассоциации обозначается сплошной линией между взаимодействующими объектами системы (рис. 8.4).

Ассоциации может быть присвоено *имя (Name)*, характеризующее природу связи. Ассоциация также может иметь *кратность (Multiplicity)*, харак-

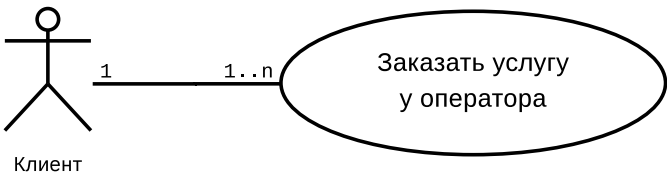


Рис. 8.4. Графическое изображение ассоциации в языке UML между актёром и вариантом использования с указанием кратности

теризующую общее количество конкретных экземпляров данного компонента, которые могут выступать в качестве элементов данной ассоциации. Кратность указывается рядом с обозначением компонента диаграммы, являющегося участником данной ассоциации.

Применительно к диаграммам вариантов использования ассоциация служит для обозначения специфической роли актёра при его взаимодействии с отдельным вариантом использования.

Отношение расширения (Extend Relationship) определяет взаимосвязь базового варианта использования с другим вариантом использования, функциональное поведение которого задействуется базовым не всегда, а только при выполнении дополнительных условий.

Отношение расширения является *отношением зависимости (Dependency Relationship)*, т.е. связью между объектами системы, при которой изменение в спецификации одного объекта может повлиять на поведение другого объекта, использующего первый объект. Зависимость изображается прерывистой линией со стрелкой, направленной к объекту, от которого имеется зависимость.

Отношение расширения между вариантами использования обозначается, как и зависимость, пунктирной линией со стрелкой, направленной от того варианта использования, который является расширением для исходного варианта использования. Данная линия со стрелкой помечается ключевым словом «extend» (рис. 8.5).

В представленном на рис. 8.5 примере при оформлении заказа услуг у оператора только в некоторых случаях может потребоваться предоставление кли-

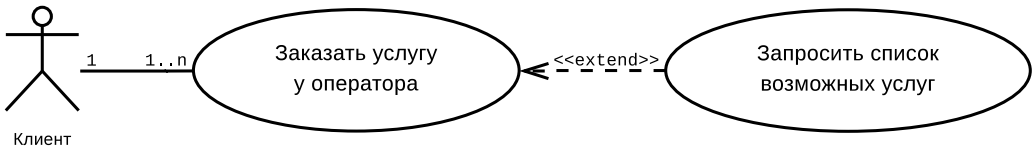


Рис. 8.5. Графическое изображение отношения расширения в языке UML

енту списка всех возможных услуг. При этом условием расширения является запрос от клиента на получение списка возможных услуг.

Отношение включения (Include Relationship) также является разновидностью отношения зависимости, но устанавливается только между двумя вариантами использования и указывает на то, что заданное поведение для одного варианта использования включается в качестве составного фрагмента в последовательность поведения другого варианта использования. Графически данное отношение обозначается как отношение зависимости в форме пунктирной линии со стрелкой, направленной от базового варианта использования к включаемому варианту использования, с указанием ключевого слова «include» (рис. 8.6).

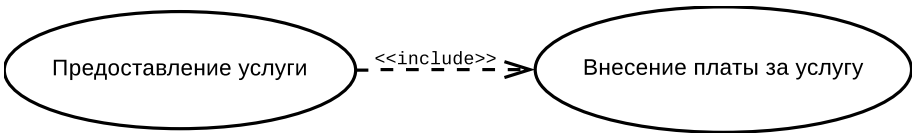


Рис. 8.6. Графическое изображение отношения включения в языке UML

Отношение обобщения (Generalization Relationship) представляет собой связь между общей сущностью, называемой родителем, и более специализированной разновидностью этой сущности, называемой потомком. Потомок наследует все свойства и поведение своего родителя, но потомок может иметь собственное поведение и свойства. В UML допускается и множественное наследование, когда один объект-потомок определяется на основе нескольких родительских объектов. Графически данное отношение обобщения обозначается сплошной линией со стрелкой в форме незакрашенного треугольника,

указывающей на родительский объект (рис. 8.7).

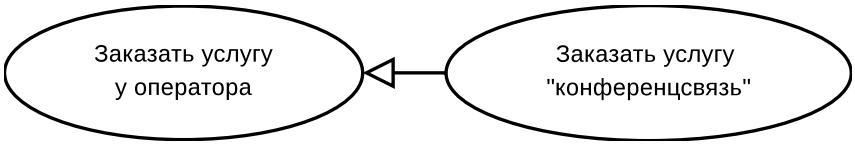


Рис. 8.7. Графическое изображение отношения обобщения в языке UML

8.4.5. Примечания

Примечания (Notes) предназначены для включения в модель произвольной текстовой информации, имеющей непосредственное отношение к контексту разрабатываемого проекта. Графически примечания обозначаются прямоугольником с «загнутым» верхним правым уголком, внутри которого содержится текст примечания. Примечание может относиться к любому элементу диаграммы, в этом случае их соединяет пунктирная линия (рис. 8.8).

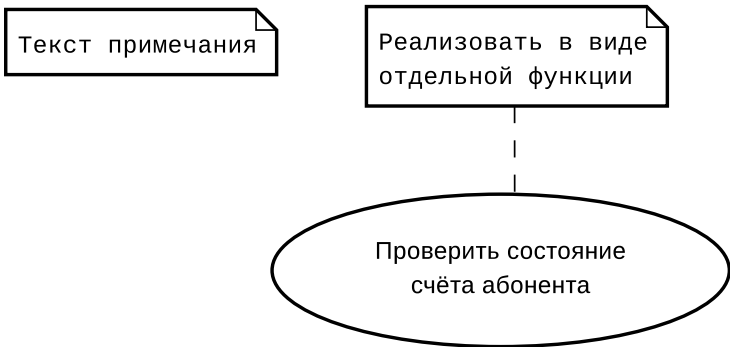


Рис. 8.8. Графическое изображение примечания в языке UML

Примечания могут присутствовать не только на диаграмме вариантов использования, но и на других канонических диаграммах.

8.4.6. Пример диаграммы вариантов использования

В качестве примера рассмотрим модель системы оказания услуг (рис. 8.9).

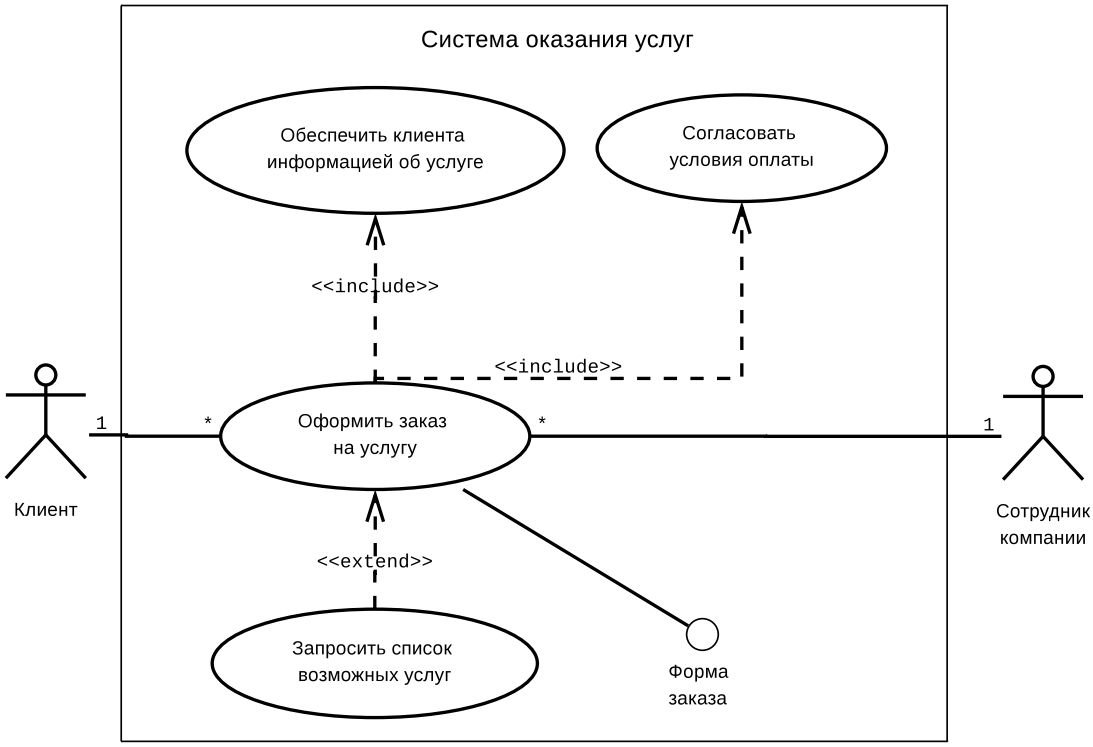


Рис. 8.9. Пример диаграммы вариантов использования

Здесь в качестве действующих лиц системы выступают два субъекта — *Клиент* и *Сотрудник компании*, оказывающей, например, услуги сотовой связи. Каждый из них взаимодействует с системой, обращаясь к сервису *Оформить заказ на услугу*, который выступает в качестве варианта использования разрабатываемой диаграммы. Значения указанных на диаграмме кратностей отражают общие правила и логику оформления заказов на услугу: один сотрудник может участвовать в оформлении нескольких заказов, но каждый заказ может быть оформлен только одним сотрудником, который несёт ответственность за корректность его оформления; аналогично каждый клиент может оформлять на себя несколько заказов, но в то же время каждый заказ должен быть оформлен на единственного покупателя.

В качестве отдельных сервисов на диаграмме выделены такие действия, как *Обеспечить клиента информацией об услуге*, *Согласовать условия оплаты*, которые раскрывают поведение исходного варианта использования, и по-

этому между ними имеет место отношение включения.

Оказание услуг предполагает наличие самостоятельного информационного объекта — списка возможных оказываемых услуг, который не зависит от реализации сервиса по обслуживанию клиентов, т.к. может запрашиваться как клиентами, так и сотрудниками компании.

Также на диаграмме присутствует объект *Форма заказа*, который выступает в качестве интерфейса взаимодействия *Клиента* и *Сотрудника компании* при оформлении заказа на услугу.

8.5. Диаграмма классов

Диаграммой классов (Class Diagram) в терминологии UML называется диаграмма, на которой показан набор классов и связей между этими классами.

Диаграмма классов не содержит информации о временных аспектах функционирования системы. Она предназначена для представления только статической структуры модели системы. В этом представлении удобнее всего описывать функциональные требования к системе — услуги, которые система предоставляет конечному пользователю.

Диаграмма классов может содержать следующие сущности: *классы*, *отношения* (*зависимости*, *обобщения*, *ассоциации*, *кооперации*), *интерфейсы*, *комментарии*, *ограничения*, *пакеты*, *подсистемы*.

8.5.1. Класс

Классом (Class) называется именованное описание совокупности объектов с общими атрибутами, операциями, связями и семантикой. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на секции, содержащие *имя класса*, *атрибуты (переменные)* и *операции (методы)* (рис. 8.10). *Имя класса* уникально в пределах пакета, указывается по центру в первой верхней секции

прямоугольника полужирным шрифтом с заглавной буквы.

Рис. 8.10. Графическое изображение класса в языке UML

Атрибутом (Attribute) называется именованное свойство класса, описывающее множество значений, которые могут принимать экземпляры этого свойства. Класс может иметь любое число атрибутов. Свойство, выражаемое атрибутом, является свойством моделируемой сущности, общим для всех объектов данного класса. Всем атрибутам должно быть присвоено некоторое значение. Имена атрибутов представляются в разделе класса, расположенном под именем класса.

Каждому атрибуту класса соответствует отдельная строка текста, которая может состоять из *квантора видимости*, *имени*, *кратности*, *типа значений* атрибута и его *исходного значения*:

```
<квантор видимости><имя>[кратность]:<тип>=<исходное значение>{свойство}
```

Квантор видимости (Visibility) может принимать одно из следующих значений:

— символ «+» — *общедоступный (Public)* — атрибут доступен или виден из любого другого класса пакета, в котором определена диаграмма;

— символ «#» — *защищённый (Protected)* — атрибут недоступен или невиден для всех классов, за исключением подклассов данного класса;

— символ «–» — *закрытый (Private)* — атрибут недоступен или невиден для всех классов без исключения;

— символ «∼» — *пакетный (Package)* — атрибут недоступен или невиден для всех классов за пределами пакета, в котором определён класс-владелец данного атрибута.

Имя атрибута представляет собой текстовую строку, которая используется в качестве идентификатора соответствующего атрибута и поэтому должна быть уникальной в пределах данного класса.

Кратность атрибута (Multiplicity) характеризует общее количество конкретных атрибутов данного типа, входящих в состав отдельного класса.

Тип атрибута представляет собой выражение, семантика которого обусловлена некоторым типом данных, определённым в пакете «Типы данных» языка UML или самим разработчиком.

Исходное значение служит для задания начального значения соответствующего атрибута в момент создания отдельного экземпляра класса.

Операцией (Operation) или методом класса называется именованная услуга, которую можно запросить у любого объекта этого класса. Совокупность операций характеризует функциональный аспект поведения класса.

Операции класса определяются в разделе, расположенном ниже раздела с атрибутами. Каждой операции класса соответствует отдельная строка текста, которая может состоять из *квантора видимости*, *имени*, *списка параметров* операции, *типа возвращаемого операцией значения*:

```
<квантор видимости><имя>(список параметров):<тип возвр.знач.>={свойство}
```

Правила задания квантора видимости и имени операции аналогичны их определению для атрибутов.

Список параметров является перечнем разделённых запятой формальных параметров, каждый из которых может быть представлен в следующем виде:

```
<вид параметра><имя параметра>:<выражение типа>=<знач. по умолчанию>,
```

где *вид параметра* — одно из ключевых слов *in*, *out* или *inout*; *имя параметра* — идентификатор соответствующего формального параметра; *выражение*

типа является зависимой от конкретного языка программирования спецификацией типа возвращаемого значения для соответствующего формального параметра; *значение по умолчанию* в общем случае представляет собой выражение для значения формального параметра, синтаксис которого зависит от конкретного языка программирования и подчиняется принятым в нём ограничениям.

Выражение типа возвращаемого значения является зависимой от языка реализации спецификацией типов значений параметров, которые возвращаются объектом после выполнения соответствующей операции.

Свойство служит для указания значений свойств, которые могут быть применены к данному элементу.

8.5.2. Отношения между классами

В диаграмме классов могут использоваться следующие типы отношений (связей): *ассоциации*, *зависимости*, *обобщения* и *реализации*. Перечисленные типы отношений уже были определены в разделе 8.4, рассматривающем построение диаграмм вариантов использования. Поэтому здесь приведены только примеры их графического изображения на диаграмме классов.

Отношение ассоциации обозначается сплошной линией с дополнительными специальными символами (имя ассоциации, имена и кратность классов-ролей ассоциации), которые характеризуют отдельные свойства конкретной ассоциации (рис. 8.11). С помощью чёрного треугольника, расположенного над линией связи справа или слева от имени ассоциации, уточняется смысл имени и указывается направление чтения имени связи.

Когда в диаграмме классов требуется отразить тот факт, что ассоциация между двумя классами имеет специальный вид «часть–целое», то используется ассоциация специального вида — *агрегатная ассоциация*. В этом случае класс «целое» имеет более высокий концептуальный уровень, чем класс «часть». Графически агрегатные ассоциации изображаются в виде простой ас-

социации с незакрашенным ромбом на стороне класса-«целого» (рис. 8.12).

Если связь «части» и «целого» настолько сильна, что уничтожение «целого» приводит к уничтожению всех его «частей», то агрегатные ассоциации, обладающие таким свойством, называются *композитными* или просто *композициями*. При наличии композиции объект-часть может быть частью только одного объекта-целого (композита). При обычной агрегатной ассоциации «часть» может одновременно принадлежать нескольким «целым». Графически композиция изображается в виде простой ассоциации, дополненной закрашенным ромбом со стороны «целого» (рис. 8.13).

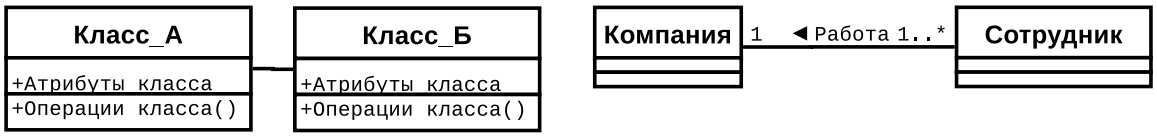


Рис. 8.11. Графическое изображение отношения ассоциации на диаграмме классов в языке UML

Рис. 8.12. Графическое изображение отношения агрегации на диаграмме классов в языке UML

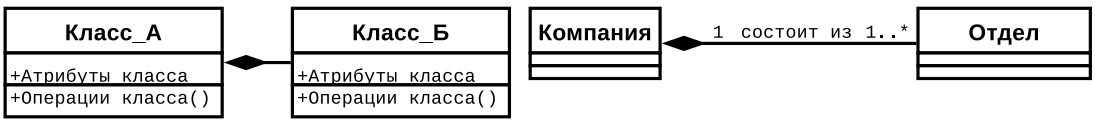


Рис. 8.13. Графическое изображение отношения композиции на диаграмме классов в языке UML

Отношение зависимости графически изображается пунктирной линией между соответствующими элементами со стрелкой, направленной от зависимого класса к независимому классу (классу-источнику) (рис. 8.14).

Рис. 8.14. Графическое изображение отношения зависимости на диаграмме классов в языке UML

Для отношения зависимости в UML предопределены ключевые слова (записываются в кавычках рядом со стрелкой), обозначающие специальные виды зависимостей:

— «access» — служит для обозначения доступности открытых атрибутов и операций класса-источника для классов-клиентов;
— «bind» — класс-клиент может использовать некоторый шаблон для своей последующей параметризации;
— «derive» — атрибуты класса-клиента могут быть вычислены по атрибутам класса-источника;
— «import» — открытые атрибуты и операции класса-источника становятся частью класса-клиента;
— «refine» — указывает, что класс-клиент служит уточнением класса-источника.

Отношение обобщения обозначается сплошной линией с треугольной стрелкой, указывающей на более общий класс (класс-предок) (рис. 8.15).

Рядом со стрелкой обобщения может размещаться строка текста (ограничение), указывающая на некоторые дополнительные свойства этого отношения:

— {complete} — означает, что в данном отношении обобщения специфицированы все классы-потомки, и других классов-потомков у данного класса-предка быть не может;

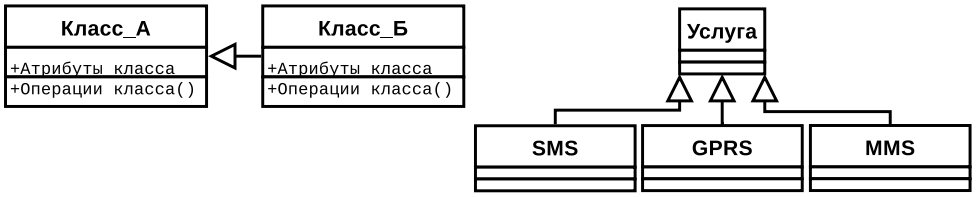


Рис. 8.15. Графическое изображение отношения обобщения на диаграмме классов в языке UML

— {disjoint} — означает, что классы-потомки не могут содержать объекты, одновременно являющиеся экземплярами двух или более классов;
— {incomplete} — означает, что на диаграмме указаны не все классы-потомки;
— {overlapping} — означает, что отдельные экземпляры классов-потомков могут принадлежать одновременно нескольким классам.

8.5.3. Интерфейс

Интерфейсы являются элементами диаграммы вариантов использования (см. раздел 8.4). При построении диаграммы классов отдельные интерфейсы могут уточняться, и в этом случае для их изображения используется специальный графический символ — прямоугольник класса с ключевым словом «interface», у которого отсутствует секция атрибутов (рис. 8.16).

Рис. 8.16. Графическое изображение интерфейса на диаграмме классов в языке UML

8.5.4. Пример диаграммы классов

На рис. 8.17 приведён пример диаграммы классов.

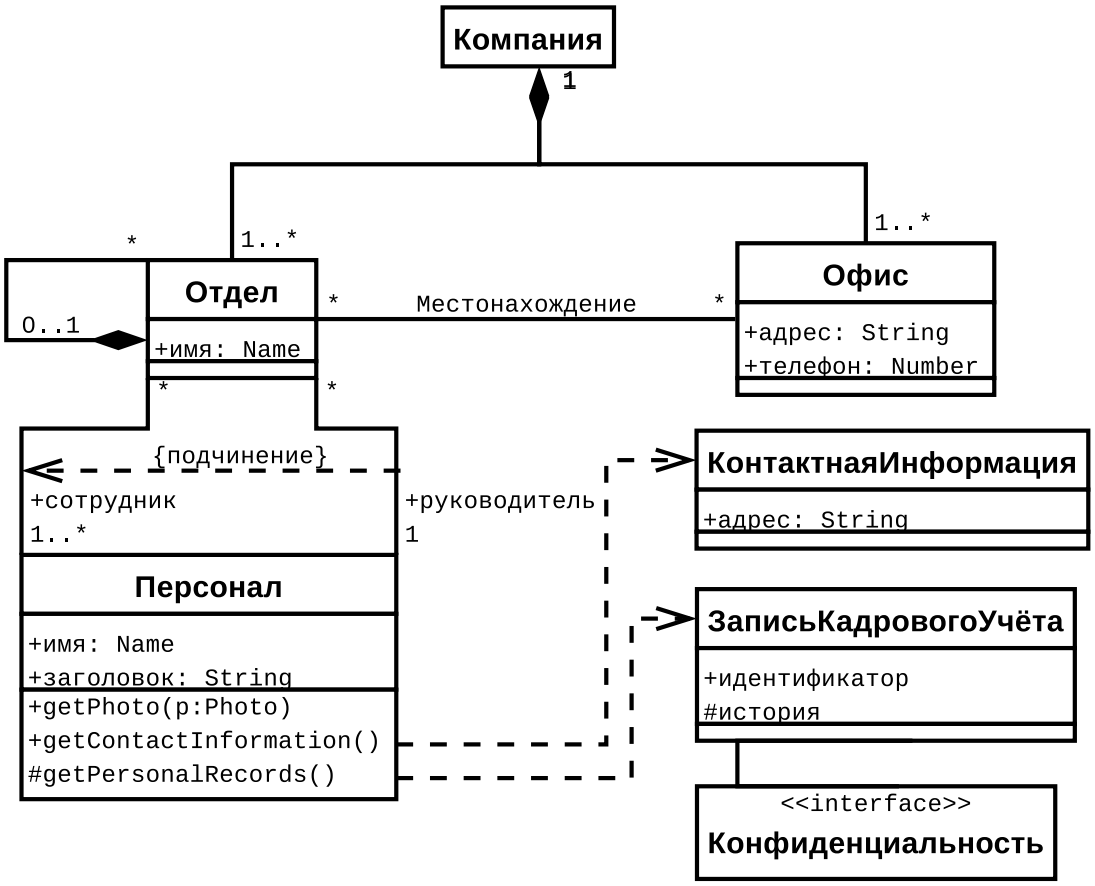


Рис. 8.17. Пример диаграммы классов UML

Компания может состоять из нескольких офисов и иметь множество отделов. При этом уничтожение класса «Компания» делает бессмысленным существование классов «Офис» и «Отдел». Поэтому классы «Офис» и «Отдел» связаны с суперклассом «Компания» композиционным отношением с указанием соответствующей кратности.

Каждый отдел может иметь название или быть безымянным, что отражено на диаграмме соответствующим, опять же композитным, отношением. Кроме того, отдел и офис ассоциируются друг с другом по месторасположению.

Класс «Отдел» ассоциируется с классом «Персонал», который определяет персонал компании — руководителей отделов и рядовых сотрудников, их отношения (подчинение). При этом персональные данные сотрудников частично зафиксированы в «Записях кадрового учёта», к которым должен быть огра-

ничен доступ, и частично в «Контактной информации», которая доступна для просмотра.

8.6. Механизмы расширения UML

Механизм расширения — один из трёх механизмов (*стереотипы*, *помеченные значения* и *ограничения*), с помощью которых можно контролируемым способом расширять язык UML.

Механизмы расширения языка UML используют, чтобы настроить язык на конкретные требования предметной области и применяемой разработчиком методики проектирования, а именно для уточнения существующих модельных элементов, переопределения стандартных компонентов UML, определения зависящих от специфики моделируемого процесса или от языка реализации программного кода расширений языка UML, присоединения произвольной семантической или несемантической информации к элементам модели.

Стереотип (Stereotype) расширяет словарь UML, позволяя создавать новые виды блоков, производные от существующих, но при этом более полно соответствующие поставленной задаче. Стереотипы обеспечивают некоторый способ классификации модельных элементов на уровне объектной модели и возможность добавления в язык UML «виртуальных» метаклассов с новыми атрибутами и семантикой.

Стереотип представляется в виде имени, заключённого в кавычки и расположенного над именем другого элемента. Стереотипный элемент можно изображать также с помощью новой связанной с ним пиктограммы.

В языке UML определён ряд стандартных стереотипов для классификаторов, компонентов, отношений и других элементов модели. В приложении А.1 приведена таблица стандартных стереотипов UML.

Помеченное значение (Tagged Value) расширяет свойства блоков UML, разрешая вносить новую информацию в спецификацию элемента. Помечен-

ные значения изображаются в виде строки в скобках, расположенной под именем другого элемента. В приложении А.2 приведена таблица стандартных помеченных значений UML.

Ограничение (Constraint) расширяет семантику блоков UML. С их помощью можно вводить новые или изменять существующие правила. Ограничения изображаются в виде строки в скобках, которая расположена возле ассоциированного элемента или связана с ним отношениями зависимости. Можно также представить ограничение в виде примечания. В приложении А.3 приведена таблица стандартных ограничений UML.

8.7. Диаграмма состояний

Диаграммы состояний (Statechart Diagram) используются для описания поведения сложных систем. Они определяют все возможные состояния, в которых может находиться объект, а также процесс смены состояний объекта в результате некоторых событий.

8.7.1. Состояние

Состояние (State) представляет собой условие или ситуацию в ходе жизненного цикла объекта, в течение которого он удовлетворяет логическому условию, выполняет определённую деятельность или ожидает события.

Событие (Event) — спецификация существенных явлений в поведении системы, которые имеют местоположение во времени и пространстве.

Состояние на диаграмме изображается прямоугольником со скруглёнными вершинами, который может быть разделен на две секции горизонтальной линией (рис. 8.18). В верхней секции записывается имя состояния, а в нижней — список некоторых внутренних действий или переходов в данном состоянии.

Имеется также два вида псевдосостояний: *начальное состояние*, в кото-

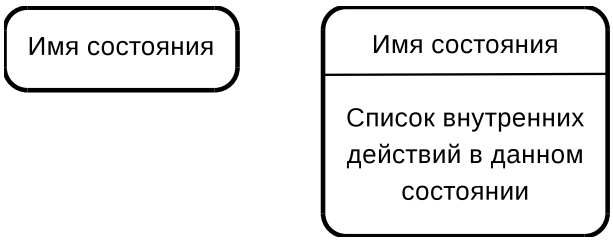


Рис. 8.18. Графическое изображение состояния в языке UML

ром находится только что созданный объект, и *конечное состояние*, которое объект не покидает, как только туда попадает (рис. 8.19).

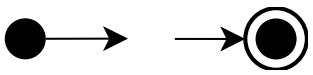


Рис. 8.19. Графическое изображение начального и конечного состояний в языке UML

8.7.2. Переход

Переход (Transition) представляет собой отношение между двумя состояниями, которое указывает на то, что объект в первом состоянии должен выполнить определённые действия и перейти во второе состояние.

На диаграмме состояний переход изображается сплошной линией со стрелкой, которая направлена в целевое состояние (рис. 8.20).

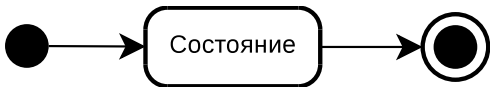


Рис. 8.20. Графическое изображение перехода из состояния в состояние в языке UML

Переходы имеют метки, которые синтаксически состоят из трёх необязательных частей — *событие*, *сторожевое условие*, *действие*:

```
<сигнатура события>[<сторожевое условие>]<выражение действия>
```

При этом сигнатура события описывает некоторое событие с необходимыми аргументами:

```
<имя события>(<список параметров, разделенных запятыми>)
```

Сторожевое условие (Guard Condition) — логическое условие, записанное в прямых скобках и представляющее собой булевское выражение, принимающее одно из двух взаимно исключающих значений: «истина» или «ложь».

Действие (Action) представляет собой спецификацию выполняемого утверждения, которая образует абстракцию вычислительной процедуры.

Действие обычно приводит к изменению состояния системы, и может быть реализовано посредством передачи сообщения объекту, модификации связи или значения атрибута.

В UML действия разделяются на *действия входа* и *действия выхода*.

Входное действие (Entry Action) — действие, которое выполняется в момент перехода в данное состояние. Обозначается с помощью ключевого слова «entry», которое указывает на то, что следующее действие должно быть выполнено в момент входа в данное состояние.

Действие выхода (Exit Action) — действие, производимое при выходе из данного состояния. Обозначается с помощью ключевого слова «exit», которое указывает на то, что следующее действие должно быть выполнено в момент выхода из данного состояния.

В UML помимо ключевых слов «entry» и «exit», определённых для действия, есть ключевое слово «do», которое определяет деятельность, выполняющуюся в течение всего времени, пока объект находится в данном состоянии, или до тех пор, пока она не будет прервана внешним событием.

8.7.3. Пример диаграммы состояний UML

Рассмотрим диаграмму состояний, которая представляет собой пример моделирования поведения конкретного объекта — процесса функционирования телефонного аппарата [12] (рис. 8.21).

Данная диаграмма состояний представляет единственный объект с одним составным состоянием. Вне этого составного состояния имеется только одно

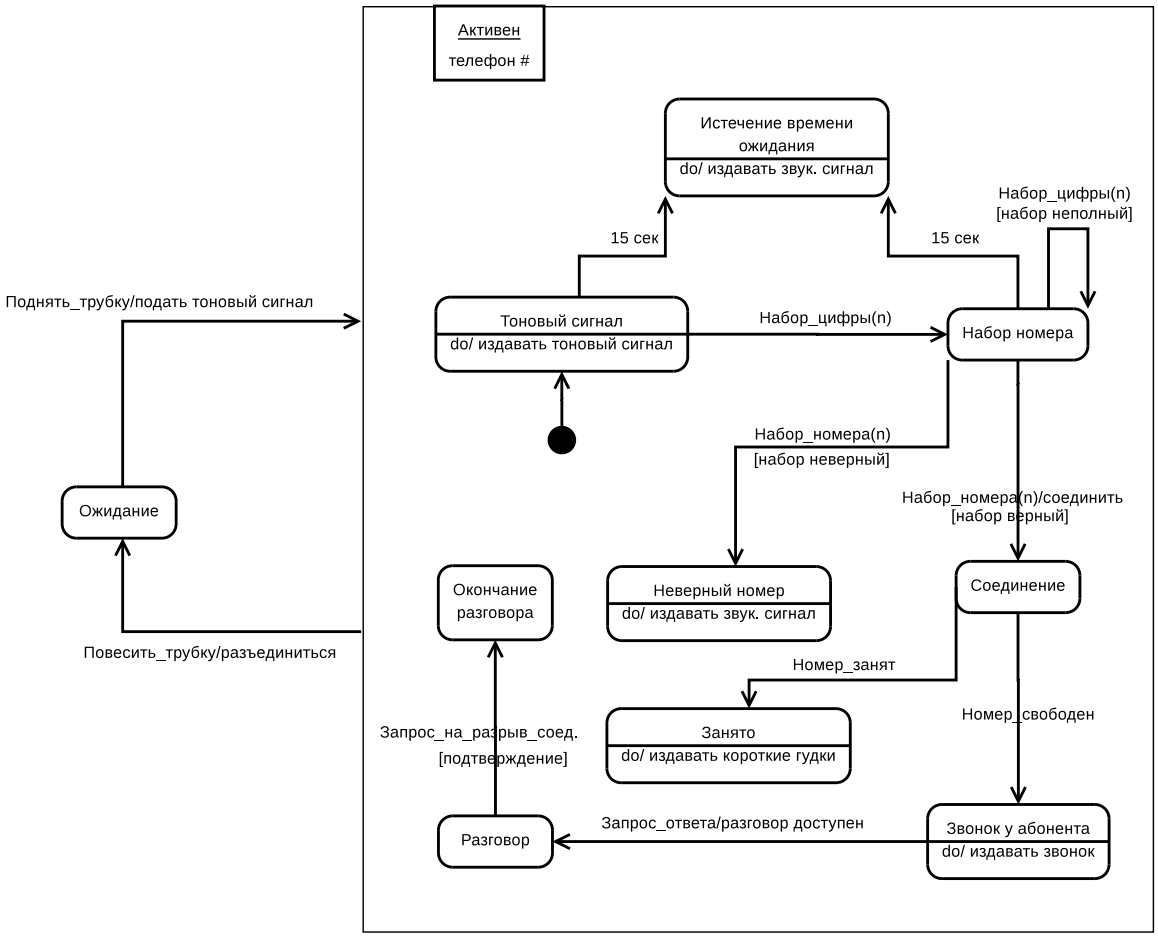


Рис. 8.21. Пример диаграммы состояний UML

состояние «Ожидание», которое характеризует исправный и подключённый к телефонной сети телефонный аппарат. Переход в следующее состояние происходит при поднятии телефонной трубки. Переход с атомарным действием «издать тоновый сигнал» переводит аппарат в составное состояние, а точнее в начальное его подсостояние.

Далее телефонный аппарат будет находиться в состоянии «Тоновый сигнал». При этом будет непрерывно издавать этот сигнал до тех пор, пока не произойдёт событие «Набор_ цифры(n)» либо не истечёт 15 секунд с момента поднятия трубки. В первом случае аппарат перейдёт в состояние «Набор номера», а во втором — в состояние «Истечение времени ожидания».

При наборе номера выполняется событие «Набор_цифры(n)» с условием

«номер неполный». Это означает, что если набранный телефонный номер не содержит необходимого количества цифр, то следует продолжить набор очередной цифры, оставаясь в состоянии «Набор номера».

Если же набранный номер полный, то можно перейти в состояние «Неверный номер» или «Соединение». В случае неверного номера (сторожевое условие «неверный» истинно) ничего не остаётся, как покинуть составное состояние, опустив трубку на рычаг. Если же номер верный, то происходит соединение по этому номеру.

Однако в результате соединения аппарат абонента может быть занят (переход в состояние «Занято») или свободен (переход в состояние «Звонок у абонента»). В первом случае можно повторить дозвон, предварительно опустив трубку на рычаг (выход из составного состояния). Во втором случае происходит проверка условия «разговор доступен». Если оно истинно, что соответствует снятию трубки абонентом, то начинается телефонный разговор. В противном случае (т.е. это условие не выполняется) телефон абонента будет продолжать звонить, извещая об отсутствии абонента или о невозможности по какой-либо причине вести разговор по телефону. При этом ничего не остаётся, как опустить трубку на рычаг.

Если же разговор состоялся, то после выполнения условия «подтверждение» на окончание разговора трубка снова опускается. При этом телефонный аппарат переходит в состояние «Ожидание», в котором может находиться неопределённо долго.

8.8. Диаграмма деятельности

Диаграмма деятельности (Activity Diagram) представляет собой диаграмму, на которой изображены переходы потока управления от одной деятельности к другой.

Деятельность (Activity) — это продолжающийся во времени неатомарный шаг вычислений в автомате. Деятельность в конечном счёте приводит к

выполнению некоего *действия (Action)*, составленного из выполняемых атомарных вычислений, каждое из которых либо изменяет состояние системы, либо возвращает какое-то значение. Действие может заключаться в вызове другой операции, посылке сигнала, создании или уничтожении объекта либо в простом вычислении.

Диаграммы деятельности относятся к динамическому аспекту поведения системы. С их помощью можно промоделировать последовательные и параллельные шаги вычислительного процесса, а также жизнь объекта, когда он переходит из одного состояния в другое в разных точках потока управления.

Графическая нотация диаграммы деятельности во многом похожа на нотацию диаграммы состояний, поскольку на ней также присутствуют обозначения состояний и переходов. Отличия следующие:

— состояния используются для представления не деятельностей, а действий (другая семантика состояний);
— на переходах отсутствует сигнатура событий.

Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении операции в предыдущем состоянии.

Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами — переходы от одного состояния действия к другому.

8.8.1. Состояния действия и состояния деятельности

Состояние действия (Action State) — это состояние, которое представляет вычисление атомарного (элементарного) действия (как правило — вызов операции). При этом такое состояние не может быть подвергнуто декомпозиции и содержать внутренние переходы.

Состояние деятельности (Activity State) — состояние, которое служит для представления процедурной последовательности действий, требующих

определённого времени выполнения.

Таким образом, состояние деятельности можно представить как составное состояние, поток управления которого включает только другие состояния деятельности и действий. Состояния деятельности могут быть подвергнуты дальнейшей декомпозиции, вследствие чего выполняемую деятельность можно представить с помощью других диаграмм деятельности. Состояния деятельности не являются атомарными и могут быть прерваны.

Графически состояния деятельности и действий обозначаются одинаково (прямоугольником с закруглёнными краями, внутри которого записывается произвольное выражение), с тем отличием, что у первого могут быть дополнительные части, такие как действия входа и выхода.

8.8.2. Переходы

Понятие *перехода (Transition)* уже было рассмотрено в разделе 8.7.2.

При построении диаграммы деятельности используются только переходы, которые срабатывают сразу после завершения деятельности или выполнения соответствующего действия. На диаграмме такой переход изображается сплошной линией со стрелкой.

Можно использовать в модели *ветвление*, которое описывает различные пути выполнения в зависимости от значения некоторого булевского выражения. При этом точка ветвления представляется ромбом. В точку ветвления может входить ровно один переход, а выходить — два или более. Для каждого исходящего перехода задаётся булевское выражение, которое вычисляется только один раз при входе в точку ветвления (рис. 8.22).

Для моделирования бизнес-процессов часто используются *параллельные потоки*. В UML для обозначения *разделения (Concurrent Fork)* и *слияния (Concurrent Join)* таких параллельных потоков выполнения используется синхронизационная черта, которая рисуется в виде жирной вертикальной или горизонтальной линии (рис. 8.23).

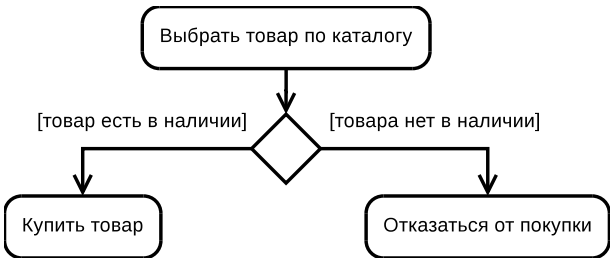


Рис. 8.22. Графическое изображение ветвления в языке UML

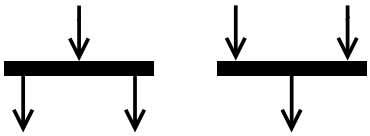


Рис. 8.23. Графическое изображение разделения и слияния параллельных потоков управления в языке UML

8.8.3. Дорожки

Дорожка (Swimlane) представляет собой графическую область диаграммы деятельности, содержащую элементы модели, ответственность за выполнение которых принадлежит отдельным подсистемам.

При моделировании течения бизнес-процессов иногда бывает полезно разбить состояния деятельности на группы, каждая из которых представляет отдел компании, отвечающий за ту или иную работу. В UML такие группы и называются *дорожками*, поскольку визуально каждая группа отделяется от соседних вертикальной чертой (рис. 8.24).

8.8.4. Объекты

В потоке управления, ассоциированном с диаграммой деятельности, могут участвовать *объекты (Objects)*, которые либо инициируют выполнение действий, либо определяют результат этих действий. При этом действия специфицируют вызовы, которые передаются от одного объекта графа деятельности другому.

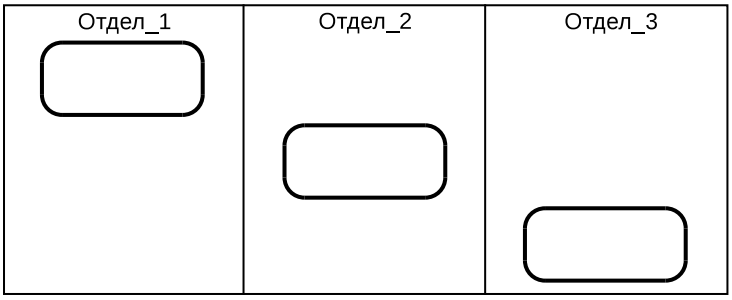


Рис. 8.24. Графическое изображение дорожек в языке UML

Графическим представлением объекта в нотации языка UML является прямоугольник класса, с тем отличием, что имя объекта подчёркивается. На диаграммах деятельности после имени может указываться характеристика состояния объекта в прямых скобках (рис. 8.25). Такие прямоугольники объектов присоединяются к состояниям действия отношением зависимости пунктирной линией со стрелкой. Соответствующая зависимость определяет состояние конкретного объекта после выполнения предшествующего действия.

Рис. 8.25. Графическое изображение объекта в языке UML

Полное имя объекта представляет собой строку текста, разделённую двоеточием и записанную в следующем формате:

```
<собственное имя объекта>/<Имя роли класса>:<Имя класса>
```

Если объект имеет *собственное имя*, то оно должно начинаться со строчной буквы. *Имя роли класса* указывается, если в модели отсутствует соответствующий класс или необходимо акцентировать внимание на особенности использования класса в рассматриваемом контексте моделирования взаимодействия. *Имя класса* — имя одного из классов диаграммы классов.

Варианты возможных записей полного имени объекта (рис. 8.26):

— o : — объект с собственным именем o;
— $o : C$ — объект с собственным именем o, являющийся экземпляром класса C;
— $: C$ — анонимный объект, являющийся экземпляром класса C;
— $o/R : C$ — объект с собственным именем o, являющийся экземпляром класса C и играющий роль R;
— $/R : C$ — анонимный объект, являющийся экземпляром класса C и играющий роль R;
— o/R — объект с собственным именем o, играющий роль R;
— $/R$ — анонимный объект, играющий роль R.

Рис. 8.26. Варианты возможных записей полного имени объекта в языке UML

В UML все объекты делятся на *пассивные* и *активные* (рис. 8.27). *Пассивный объект* оперирует только данными и не может инициировать деятельность по управлению другими объектами, но может посылать сигналы в процессе выполнения запросов. У *активного объекта* есть собственный процесс управления и он может инициировать деятельность по управлению другими объектами (обозначается прямоугольником с утолщёнными границами).

Относящиеся к деятельности объекты можно включить в диаграмму деятельности, и с помощью символа зависимости привязать к той деятельности или переходу, где они создаются, модифицируются или уничтожаются. Такое сочетание зависимостей и объекта называется *траекторией объекта (Object Flow)*, поскольку описывает его участие в потоке управления.

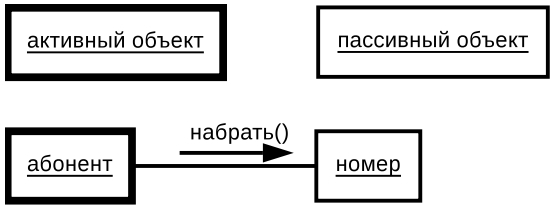


Рис. 8.27. Активный и пассивный объекты в языке UML

На диаграмме деятельности с дорожками расположение объекта может иметь дополнительный смысл. А именно, если объект расположен на границе двух дорожек, то это может означать, что переход к следующему состоянию действия в соседней дорожке ассоциирован с нахождением объекта в некотором состоянии. Если же объект расположен внутри дорожки, то и состояние этого объекта целиком определяется действиями данной дорожки.

Диаграмму деятельности можно присоединить к любому элементу модели для визуализации, специфицирования, конструирования и документирования поведения этого элемента, в частности, к классам, интерфейсам, компонентам, узлам, прецедентам и кооперациям. Чаще всего диаграммы деятельности присоединяются к операциям.

8.8.5. Пример диаграммы деятельности

В качестве примера можно рассмотреть фрагмент диаграммы деятельности торговой компании, обслуживающей клиентов в форме заказов [13, 12] (рис. 8.28).

Подразделениями компании обычно являются отдел приёма и оформления заказов, отдел продаж и склад. Этим подразделениям будут соответствовать три дорожки на диаграмме деятельности, каждая из которых специфицирует зону ответственности подразделения. В этом случае диаграмма деятельности заключает в себе не только информацию о последовательности выполнения рабочих действий, но и о том, какое подразделение торговой компании должно выполнять то или иное действие.

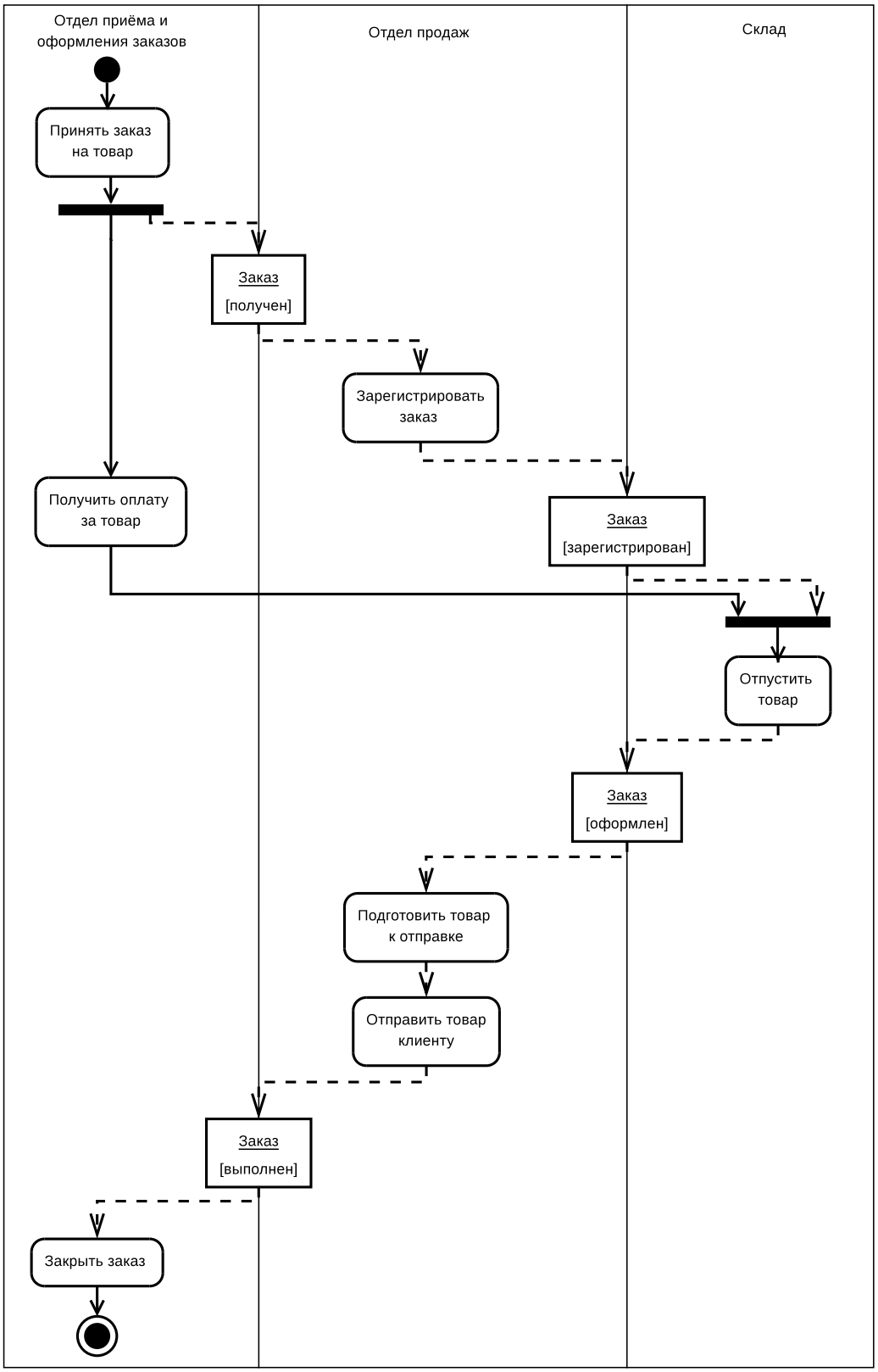


Рис. 8.28. Пример диаграммы деятельности

Отделом приёма и оформления заказов фиксируется полученный от клиента заказ, после чего он регистрируется в отделе продаж. Затем осуществляется распараллеливание деятельности на два потока (переход-разделение). Первый из них остаётся в этом же отделе и связан с получением оплаты от клиента за заказанный товар. Второй инициирует выполнение действия по регистрации заказа в отделе продаж (модель товара, размеры, цвет, год выпуска и пр.) и передаче товара на склад.

Однако выдача товара со склада начинается только после того, как от клиента будет получена оплата за товар (переход-слияние). Затем выполняется подготовка товара к отправке и его отправка клиенту в отделе продаж. После завершения этих деятельностей заказ закрывается в отделе приёма и оформления заказов.

8.9. Диаграмма взаимодействия

Диаграммой взаимодействия (Interaction Diagram) называется диаграмма, на которой представлено взаимодействие, состоящее из множества объектов и отношений между ними, включая и сообщения, которыми они обмениваются.

Диаграммы взаимодействия относятся к динамическому виду системы, причём их можно строить двумя способами: упорядочивая по времени сообщения или структурно упорядочивая взаимодействующие объекты. Получаемые любым из этих способов диаграммы семантически эквивалентны и преобразуются друг в друга без потери информации. По аспекту взаимодействия диаграммы взаимодействия делятся на *диаграммы последовательностей* и *диаграммы кооперации*.

8.9.1. Диаграмма последовательностей

Диаграммой последовательностей (Sequence Diagram) называется диаграмма взаимодействий, акцентирующая внимание на временной упорядоченности сообщений, которыми обмениваются объекты системы.

Графически диаграмма последовательностей представляет собой таблицу, объекты в которой располагаются вдоль оси X, а сообщения в порядке возрастания времени — вдоль оси Y. На диаграмме изображаются только объекты, непосредственно участвующие во взаимодействии.

Диаграммы последовательностей характеризуются двумя особенностями, отличающими их от диаграмм кооперации: *линией жизни объекта* и *фокусом управления*.

Линия жизни объекта (Object Lifeline) представляет собой вертикальную пунктирную линию, отражающую существование объекта во времени (рис. 8.29). При этом каждый объект графически изображается в форме прямоугольника и располагается в верхней части своей линии жизни.

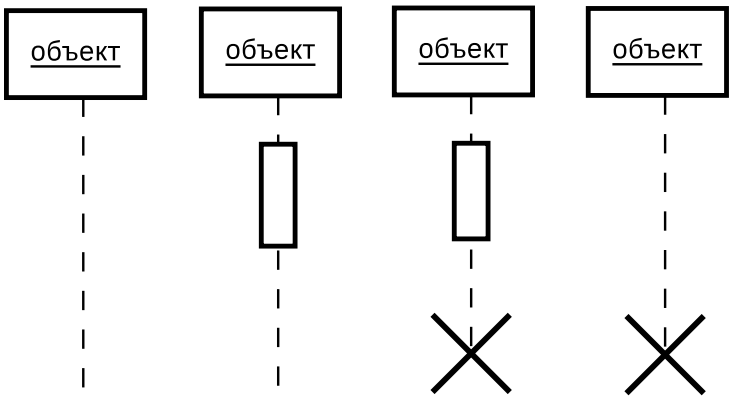


Рис. 8.29. Графическое изображение различных вариантов линий жизни и фокусов управления объектов в языке UML

Фокус управления (Focus of Control) изображается в виде вытянутого прямоугольника, показывающего промежуток времени, в течение которого объект выполняет какое-либо действие непосредственно или с помощью подчи-

нённой процедуры. Верхняя грань прямоугольника выравнивается по временной оси с моментом начала действия, нижняя — с моментом его завершения.

Отдельные объекты, выполнив свою роль в системе, могут быть уничтожены, чтобы освободить занимаемые ими ресурсы. Для обозначения момента уничтожения объекта в языке UML используется специальный символ в форме латинской буквы «X» (рис. 8.29).

Взаимодействие между объектами описывается совокупностью сообщений, которыми эти объекты обмениваются между собой. *Сообщение (Message)* представляет собой законченный фрагмент информации, который инициирует выполнение определённых *действий*, направленных на решение некоторой задачи тем объектом, которому это сообщение отправлено.

В UML определены следующие виды действий:

— «call» (вызвать) — вызывает операцию, применяемую к объекту;
— «return» (возвратить) — возвращает значение вызывающему объекту;
— «send» (послать) — посылает объекту сигнал;
— «create» (создать) — создаёт новый объект;
— «destroy» (уничтожить) — удаляет объект.

В UML различают следующие сообщения (рис. 8.30):

— сообщения для вызова процедур, выполнения операций или обозначения отдельных вложенных потоков управления — всегда выходит из фокуса управления или линии жизни объекта, инициирующего сообщение, и стрелкой соприкасается с линией жизни объекта, которому предназначено сообщение, с возможной передачей фокуса управления (графически — сплошная линия с закрашенной стрелкой);
— сообщения для обозначения простого асинхронного сообщения, передаваемого в произвольный момент времени (графически — сплошная линия со стрелкой);
— сообщения для возврата из вызова процедуры (графически — пунктирная линия со стрелкой или без неё).

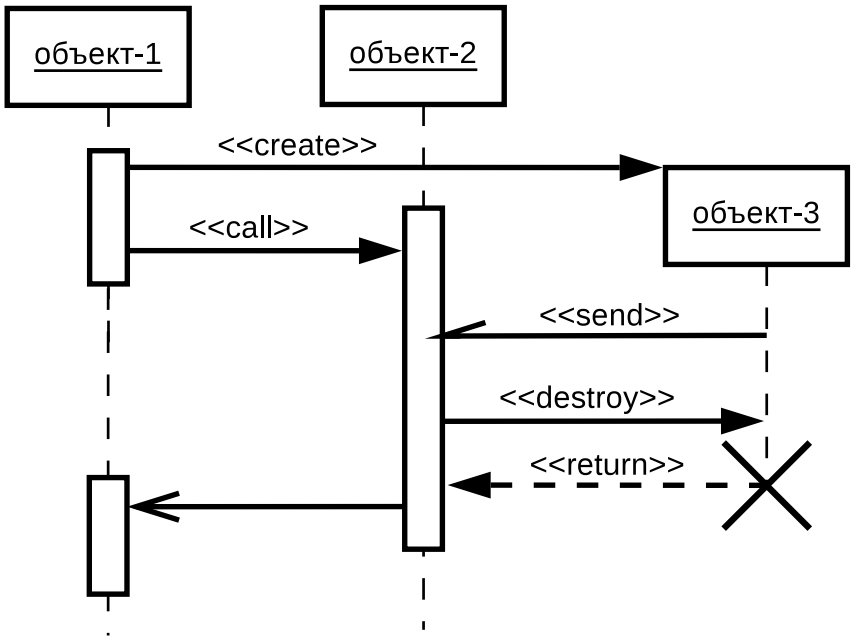


Рис. 8.30. Графическое изображение сообщений, передаваемых между объектами на диаграмме последовательностей в языке UML

На диаграмме последовательностей все сообщения упорядочены по времени своего возникновения в моделируемой системе. Сообщения изображаются в виде горизонтальных стрелок с именем сообщения (или без него) и образуют определённый порядок относительно времени своей инициализации.

8.9.2. Диаграмма кооперации

Диаграммой кооперации (Collaboration Diagram) называется диаграмма взаимодействий, основное внимание в которой уделяется структурной организации объектов, принимающих и отправляющих сообщения. Графически такая диаграмма представляет собой граф из вершин и рёбер.

Кооперация (Collaboration) представляет собой спецификацию множества объектов отдельных классов, совместно взаимодействующих с целью реализации отдельных вариантов использования в общем контексте моделируемой системы. Кооперация определяет структуру поведения системы в терминах

взаимодействия участников этой кооперации.

Понятие объекта в языке UML было рассмотрено в разделе 8.8.4.

Для создания диаграммы кооперации нужно расположить участвующие во взаимодействии объекты в виде вершин графа. Затем связи, соединяющие эти объекты, изображаются в виде дуг этого графа. Связи дополняются сообщениями, которые объекты принимают и посылают.

Диаграммы кооперации характеризуются двумя особенностями, отличающими их от диаграмм последовательностей: *путём (связью)* и *порядковым номером сообщения*.

Путь или связь (Link) используется для описания связи одного объекта с другим. Связи не имеют собственных имён и кратности, но могут использоваться следующие стереотипы:

— «association» — указывает, что связь является ассоциацией;
— «parameter» — указывает, что связь является параметром некоторого метода;
— «local» — указывает, что связь является локальной переменной метода, область видимости которой ограничена только соседним объектом;
— «global» — указывает, что связь является глобальной переменной, область видимости которой распространяется на всю диаграмму кооперации;
— «self» — указывает, что связь является рефлексивной связью объекта с самим собой (изображается петлёй в верхней части прямоугольника объекта).

Порядковый номер сообщения используется для обозначения временной последовательности сообщений.

8.9.3. Пример диаграммы взаимодействия

На рис. 8.31 приведён пример диаграммы взаимодействия «Телефонный разговор».

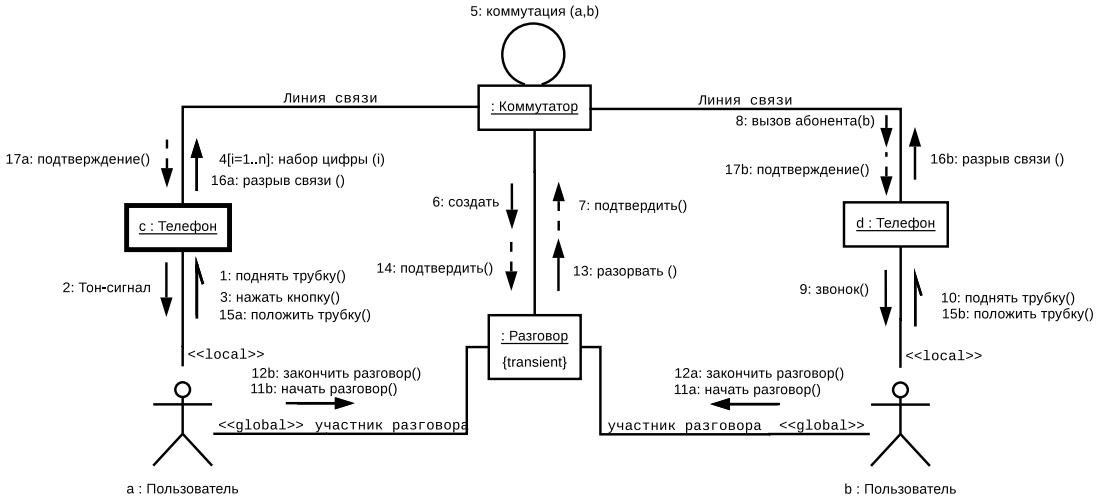


Рис. 8.31. Пример диаграммы взаимодействия «Телефонный разговор»

Объектами в этом примере являются два абонента a и b, два телефонных аппарата c и d, коммутатор и сам разговор как объект моделирования. При этом как коммутатор, так и разговор являются анонимными объектами, первый телефонный аппарат изображён как активный объект, а второй — как пассивный.

Все связи на диаграмме специфицированы, на их концах указана информация в форме ролей связей. Заметим, что для объекта «Разговор» указано помеченное значение `{transient}`, которое означает, что этот объект создаётся в процессе выполнения объемлющего процесса и уничтожается до его завершения.

Для того чтобы отобразить структурную организацию потоков управления, на диаграмме изображены все сообщения с указанием их порядка и семантических особенностей.

8.10. Диаграмма реализации UML

Для создания конкретной физической системы необходимо реализовать все элементы логического представления в конкретные материальные сущно-

сти. Для описания таких реальных сущностей используется физическое представление модели. В контексте языка UML это означает совокупность связанных физических сущностей, включая программное и аппаратное обеспечение, а также персонал, которые организованы для выполнения специальных задач.

Физическая система (Physical System) — реально существующий прототип модели системы.

Базовыми элементами физического представления системы в нотации языка UML являются исполняемые модули, библиотеки классов и процедур, стандартных графических интерфейсов, файлов баз данных, которые в совокупности составляют программный код системы. При этом программная система может считаться реализованной, если она будет способна выполнять функции своего целевого предназначения.

Полный проект программной системы представляет собой совокупность моделей логического и физического представлений, которые должны быть согласованы между собой. В языке UML для физического представления моделей систем используются так называемые *диаграммы реализации*, которые включают в себя две отдельные канонические диаграммы: *диаграмму компонентов* и *диаграмму развёртывания*.

8.10.1. Диаграмма компонентов

Диаграмма компонентов — диаграмма, на которой изображена организация некоторого множества компонентов и зависимости между ними.

Диаграмма компонентов используется для моделирования статического вида системы с точки зрения реализации, т.е. описывает особенности физического представления системы. Этот тип диаграмм позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых могут выступать исходный, бинарный и исполняемый коды.

Диаграммы компонентов обычно включают в себя:

— компоненты;

— интерфейсы;

— отношения зависимости, обобщения, ассоциации и реализации.

Компонент (Component) — физически существующая часть системы, которая обеспечивает реализацию классов и отношений, а также функциональное поведение моделируемой программной системы. Компонентом может быть исполняемый код отдельного модуля, командные файлы или файлы, содержащие интерпретируемые скрипты.

Для графического представления компонента используется специальный символ — прямоугольник со вставленными слева двумя более мелкими прямоугольниками, внутри которого записывается имя компонента и, возможно, дополнительная информация (рис. 8.32).

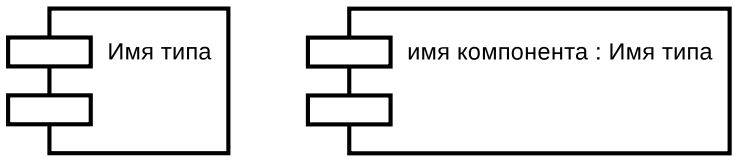


Рис. 8.32. Графическое изображение компонента в языке UML

Имя компонента подчиняется общим правилам именования элементов модели в языке UML и может состоять из любого числа букв, цифр и некоторых знаков препинания. При этом, если компонент представляется на уровне типа, то в качестве его имени записывается только имя типа с заглавной буквы, и если компонент представляется на уровне экземпляра, то в качестве его имени записывается `<имя компонента> : <Имя типа>`.

В языке UML определены следующие виды компонентов:

— компоненты развёртывания — обеспечивают выполнение функций системы (например, динамически подключаемые библиотеки, файлы справки и т.п.);

— компоненты «рабочие продукты» (файлы с исходными текстами программ);
— компоненты исполнения (исполняемые модули).

В языке UML для компонентов определены следующие стереотипы:

— «library» (библиотека) — компонент в форме динамической или статической библиотеки;
— «table» (таблица) — компонент в форме таблицы базы данных;
— «file» (файл) — компонент в виде файла с исходными текстами программ;
— «document» (документ) — компонент в форме документа;
— «executable» (исполняемый) — компонент, который может исполняться в узле.

Понятие интерфейса рассматривалось в разделах 8.4.3 и 8.5.3. Поэтому здесь отметим те особенности интерфейсов, которые характерны для их представления на диаграммах компонентов.

Интерфейс графически изображается окружностью или в виде прямоугольника класса со стереотипом «interface» и секцией поддерживаемых операций (рис. 8.32). Соединяется интерфейс с компонентом отрезком линии без стрелок, что семантически означает реализацию интерфейса. Наличие интерфейсов у компонента означает, что данный компонент реализует соответствующий набор интерфейсов.

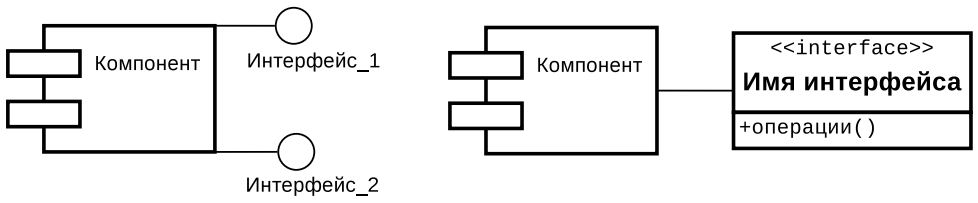


Рис. 8.33. Графическое изображение интерфейса в языке UML

Различают два способа связи интерфейса и компонента. Если компонент реализует некоторый интерфейс, то такой интерфейс называют *экспортиру-*

емым (или поддерживаемым), поскольку этот компонент предоставляет его в качестве сервиса другим компонентам. Если же компонент использует некоторый интерфейс, реализуемый другим компонентом, то такой интерфейс для первого компонента называется *импортируемым*. Особенность импортируемого интерфейса состоит в том, что на диаграмме компонентов это отношение изображается с помощью зависимости.

Отношения зависимости, обобщения, ассоциации и реализации также рассматривались ранее в разделах 8.4.4 и 8.5.2. Поэтому отметим только то, что *отношения зависимости*, применительно к диаграмме компонентов, могут связывать компоненты и импортируемые ими интерфейсы, а также различные виды компонентов между собой, что изображается пунктирной линией со стрелкой, направленной от клиента или зависимого элемента к источнику или независимому элементу модели (рис. 8.34).

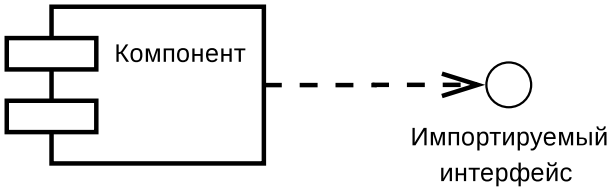


Рис. 8.34. Графическое изображение отношения зависимости на диаграмме компонентов в языке UML

8.10.2. Диаграмма развёртывания

Диаграмма развёртывания (Deployment Diagram) — диаграмма, на которой представлена конфигурация обрабатывающих узлов и размещённые на них компоненты.

Диаграмма развёртывания применяется для представления общей конфигурации и топологии распределённой программной системы и содержит изображение размещения компонентов по отдельным узлам системы. Кроме того, она показывает наличие физических соединений — маршрутов передачи ин-

формации между аппаратными устройствами, задействованными в реализации системы.

Диаграмма развёртывания содержит графические изображения процессоров, устройств, процессов и связей между ними.

Узел (Node) представляет собой физически существующий элемент системы, который может обладать вычислительным ресурсом или являться техническим устройством.

Графически узел на диаграмме развёртывания изображается в форме куба (рис. 8.35). Узел имеет имя, которое указывается внутри этого графического символа.

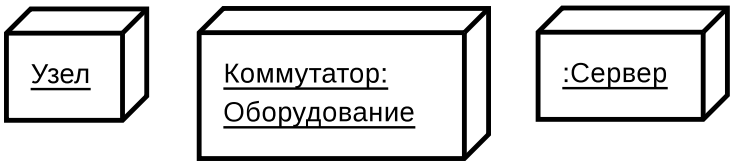


Рис. 8.35. Графическое изображение узла в языке UML

На диаграмме развёртывания кроме изображения узлов указываются *отношения* между ними (рис. 8.36). В качестве отношений выступают физические соединения между узлами, а также зависимости между узлами и компонентами (рис. 8.37).

Соединения являются разновидностью ассоциации и изображаются отрезками линий без стрелок. Наличие такой линии указывает на необходимость организации физического канала для обмена информацией между соответствующими узлами. Характер соединения может быть дополнительно специфицирован примечанием, стереотипом, помеченным значением или ограничением.

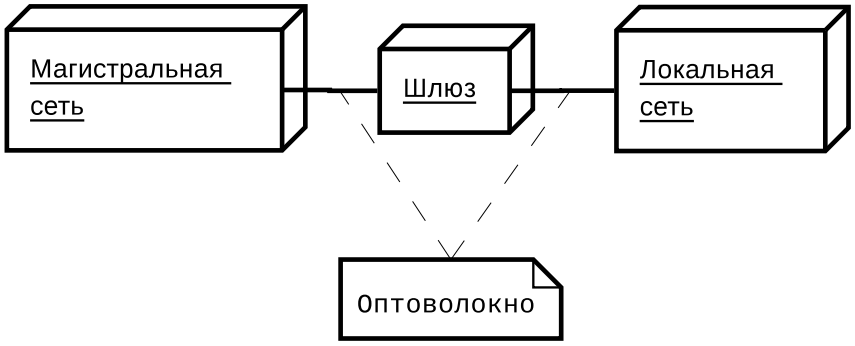


Рис. 8.36. Графическое изображение отношений между узлами в языке UML

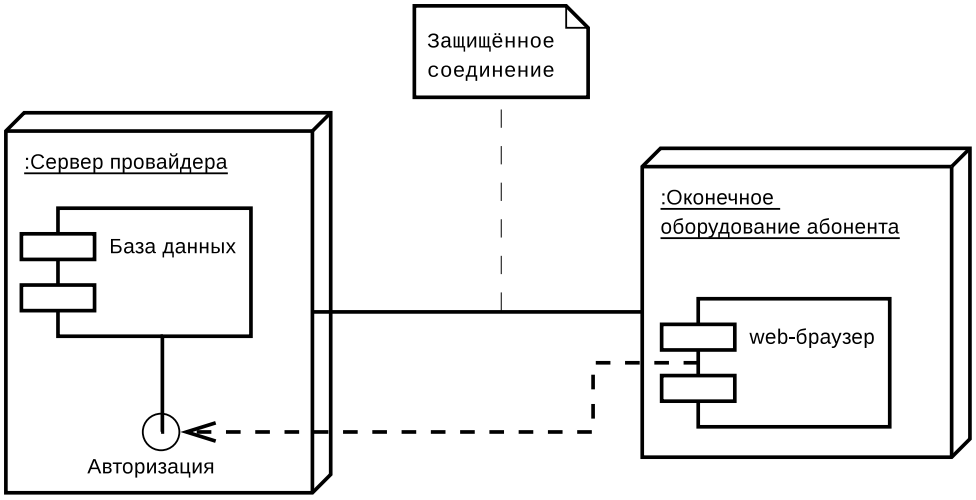


Рис. 8.37. Графическое изображение связи и зависимости между узлами и компонентами в языке UML

Глава 9. Описание бизнес-процессов с использованием обозначений BPMN. Язык моделирования бизнес-процессов BPML

9.1. Нотация BPMN

Основным инструментом BPMN служит *диаграмма бизнес-процессов (Business Process Diagram, BPD)*. Полученная в результате модель представляет собой сеть графических объектов, которые изображают действия, связанные потоками управления.

9.1.1. Типы процессов в нотации BPMN

В рамках общей нотации BPMN существует три типа процессов:

— *частные, или внутренние, процессы (Private)* — внутренние процессы определённой организации;

— *абстрактные, или открытые, процессы (Abstract)* — взаимодействие между процессами на уровне обмена сообщениями;

— *совместные, или глобальные, процессы (Collaboration)* — два и более абстрактных процесса на одной диаграмме.

Частные бизнес-процессы обычно называют *WorkFlow или процессами BPM* (управление деловыми процессами). На диаграммах BPMN каждый частный бизнес-процесс помещается в отдельную область, и таким образом последовательный поток процесса содержится внутри области и не может пересекать её границы. При этом поток сообщений может пересекать границы области с целью указания на взаимодействия, существующие между отдельными частными бизнес-процессами.

Абстрактными считаются процессы, действия которых имеют связи за пределами частного бизнес-процесса. Кроме того, к абстрактным процессам относятся соответствующие механизмы контроля потока. Абстрактные процессы содержатся внутри области и могут моделироваться отдельно или внутри общей схемы BPMN для демонстрации потока сообщений между блоками абстрактного процесса и другими объектами.

Совместный процесс отображает взаимодействие между двумя и более бизнес-объектами. Его можно изобразить в виде двух или более взаимодействующих абстрактных процессов.

9.1.2. Точки зрения

Диаграмма BPMN может отображать процессы разных участников, каждый из которых может иметь свой взгляд на неё. По отношению к участнику одни действия будут внутренними, а другие — внешними. Во время выполнения процесса разница между внутренними и внешними действиями имеет большое значение для определения участником статуса действия или для поиска проблем. Однако сама схема останется неизменной. Поэтому проектирование процессов рекомендуется разделять по следующим уровням:

— *бизнес-уровень (Business Layer)* — общее представление различных бизнес-шагов и управляющих ими потоков;

— *функциональный уровень (Functional Layer)* — общее представление о взаимодействии процессов с базами данных, проектирование формата обмена сообщениями;

— *уровень реализации (Implementation Layer)* — схема реализации деталей процесса.

Все уровни зависимы между собой, и просто предлагают различное представление одного и того же процесса.

9.1.3. Объекты потока процесса

В BPMN определены следующие типы объектов:

— *деятельность (Activity)* — действия, выполняемые участниками процесса;
— *соединитель потоков (Flow Connector)* — связь между объектами процесса;
— *событие (Event)* — спецификация существенных явлений в поведении системы (т.е. явления, влияющие на поток процесса);
— *шлюз или объединение (Gateway)* — точка принятия решений в диаграмме процесса, после которой поток процесса может быть продолжен по одному или более путям;
— *дорожка (Swimlane)* — область диаграммы, содержащая элементы модели отдельного участника процесса;
— *артефакт (Artifact)* — документы и комментарии.

Деятельность состоит из атомарных или составных действий, характеризующих работу, которую выполняет система. Деятельность подразделяется на *задачи* и *подпроцессы*.

Задача (Task) представляет собой элементарное действие в пределах процесса. Задачи могут быть простыми и повторяющимися. Графически задачи изображаются в виде прямоугольника со скруглёнными углами (рис. 9.1), могут содержать *метку (Label)*, *документацию (Documentation)*, связанную с задачей, *тип повторения задачи (Loop Type)*.

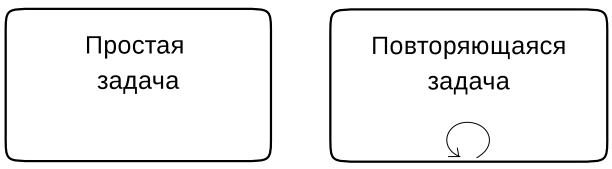


Рис. 9.1. Графическое изображение задачи в нотации BPMN

Задачи могут объединяться в *подпроцессы (Sub-process)* — сложные действия в пределах процесса. Подпроцессы могут быть обычными и повторяю-

щимися, сжатыми и расширенными (рис. 9.2).

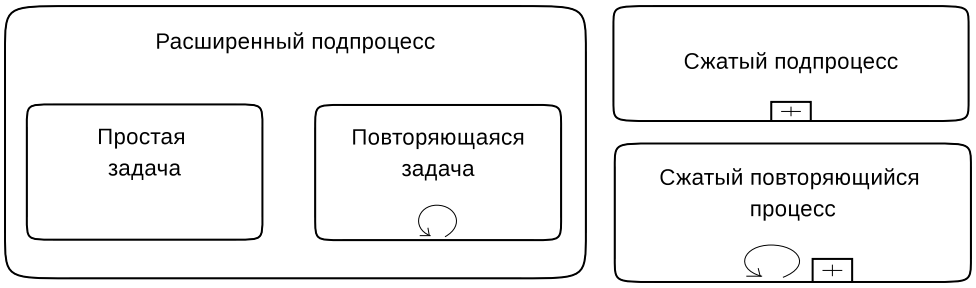
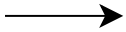
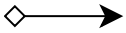
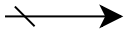
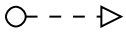
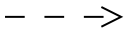


Рис. 9.2. Графическое изображение подпроцесса в нотации BPMN

Соединители потоков (коннекторы) используются для соединения элементов потока процесса (табл. 9.1).

Таблица 9.1

Типы коннекторов в нотации BPMN

Графический символ	**Название элемента**
	простой поток
	условный поток
	поток по-умолчанию
	поток сообщений
	ассоциация

Последовательный (простой) поток (Sequence Flow) используется для отображения порядка следования действий процесса. Графически последовательный поток изображается сплошной линией с закрашенной стрелкой.

Условный поток — последовательный поток с условным выражением, измеряемым по времени выполнения, с целью определить, будет ли использоваться поток. Графически условный поток изображается сплошной линией с закрашенной стрелкой и ромбиком на противоположном конце.

Поток по-умолчанию — условный поток, который будет использоваться в случае, если все другие условные потоки не верны при выполнении. Графи-

чески поток по-умолчанию изображается сплошной линией с закрашенной стрелкой и косой чертой на противоположном конце.

Поток сообщений (Message Flow) используется для отображения потока сообщений между двумя отдельными участниками процесса. Графически поток сообщений изображается пунктирной линией с незакрашенной стрелкой и незакрашенным кружочком на противоположном конце.

Ассоциация (Association) используется для того, чтобы связать данные, текст и другие артефакты с потоком объектов процесса. Графически ассоциация изображается пунктирной линией с V-образной стрелкой.

Событие представляет собой нечто, происходящее в ходе бизнес-процесса и влияющее на его течение. Так, указание типа триггера (условия или ограничения) на событие устанавливает определённые ограничения на процесс потока. Также события могут приводить бизнес-процесс к некоторому результату.

Существует три типа событий (рис. 9.3), классифицированных по времени воздействия на ход процесса: *начальные (Start Events)*, *промежуточные (Intermeidate Events)* и *конечные (End Events)*. Начальные и конечные события представляют собой точки начала и окончания процесса и должны обязательно присутствовать на диаграмме.

Рис. 9.3. Графическое изображение начального, промежуточного и конечного событий в нотации BPMN

В нотации BPMN определены следующие типы триггеров (табл. 9.2):

— *сообщение (Message)* — исходит от некоторого участника или триггера процесса и предшествует началу, продолжению или окончанию некоторого действия процесса;

— *таймер (Timer)* — устанавливает цикл времени течения процесса;
— *правило (Rule)* — тестовая строка, описывающая некоторое правило, применяемое к событию;
— *исключительное событие (Exception)* — при завершении некоторого действия информирует процесс о возникновении ошибки;
— *компенсация (Compensation)* — показывает, как подпроцесс может быть скомпенсирован последовательностью отката;
— *отмена (Cancel)* — указывает на отмену события;
— *ссылка (Link)* — представляет собой механизм, обеспечивающий подключение окончания события одного потока процесса к началу события другого потока процесса;
— *составное событие (Multiple)* — указывает на то, что событие может задействовать несколько путей развития процесса или продолжить процесс в случае наличия промежуточного события.

Таблица 9.2

Типы триггеров в нотации BPMN

Графический символ	**Название элемента**
	события с триггером «сообщение»
	события с триггером «таймер»
	события с триггером «правило»
	события с триггером «исключительное событие»
	события с триггером «компенсация»
	события с триггером «отмена»
	события с триггером «ссылка»
	события с триггером «составное событие»

Шлюз (или объединение) используется для контроля расхождения и схождения последовательного потока и обозначает ветвление или соединение маршрутов (табл. 9.3). Внутренние маркеры указывают на тип контроля развития процесса. Шлюзы могут определять направление потока *на основе данных процесса (Data-Based)* или *на основе результатов наступления событий (Event-Based)*.

Таблица 9.3

Типы шлюзов в нотации BPMN

Граф. символ	**Название элемента**
	шлюз
	шлюз на основе данных процесса с оп. XOR
	шлюз на основе результатов наступления событий с оп. XOR
	шлюз на основе результатов наступления событий с оп. OR
	шлюз с оп. AND
	шлюз со сложным условием

В нотации BPMN определены следующие виды шлюзов:

— *шлюз на основе данных процесса с операцией «исключающее ИЛИ» (Exclusive (XOR) Data-Based)* — может выполняться только одна из ветвей процесса;

— *шлюз на основе результатов наступления событий с операцией «исключающее ИЛИ» (Exclusive (XOR) Event-Based)* — может выполняться только одна из ветвей процесса;

— *шлюз на основе результатов наступления событий с операцией «ИЛИ» (Inclusive (OR) Event-Based)* — могут выполняться одна или более ветвей процесса;

— *шлюз с операцией «И» (Parallel (AND))* — все ветви процесса выполняются параллельно;
— *шлюз со сложным условием (Complex)*.

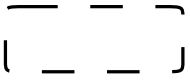
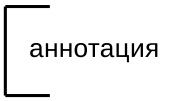
Дорожки представляют собой участников процесса и группируют процесс по ответственности и категориям исполнителей (рис. 9.4).

Рис. 9.4. Графическое изображение дорожек в нотации BPMN

В BPMN определены следующие типы *артефактов* (табл. 9.4):
— данные об объекте (Data Objects) — представляют собой дополнительные данные об объекте, графически изображаются в виде прямоугольника с «загнутым» верхним правым уголком;
— группа (Groupe) — используется для документации или анализа целей, но не влияет на последовательность потоков;
— аннотация (Annotation) — предоставляет дополнительную информацию для читателя диаграмм BPMN.

Таблица 9.4

Артефакты в нотации BPMN

Графический символ	**Название элемента**
	данные об объекте
	группа
	аннотация

9.1.4. Пример диаграммы BPMN

На рис. 9.5 приведён пример BPMN-процесса «Доставка товара в магазин». Магазин отправляет заявку на товар дистрибьютору. Дистрибьютор подтверждает получение заявки, запрашивает товар со склада. Перед отправкой товара со склада проверяется наличие товара, при необходимости товар заказывается у поставщика, после чего товар доставляется грузоперевозчиком в магазин.

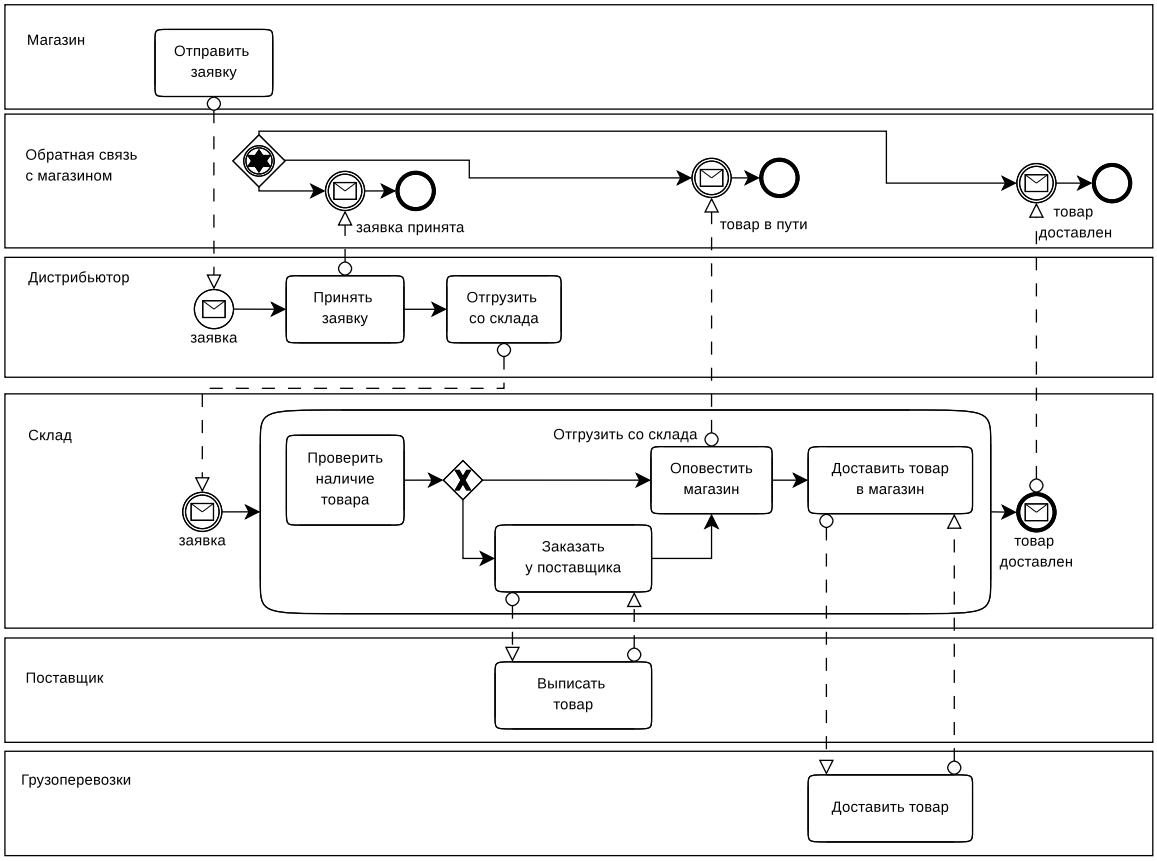


Рис. 9.5. Пример BPMN процесса «Доставка товара в магазин»

9.2. Язык моделирования бизнес-процессов BPML

Язык моделирования бизнес-процессов (Business Process Modeling Language, BPML) базируется на метаязыке XML. Бизнес-процесс в BPML соот-

ветствует иерархическому набору вложенных и последовательных тегов.

BPML предназначен для представления формальной модели, описывающей процессы, выполняемые в системе, которые определяют все аспекты корпоративных бизнес-процессов. BPML задаёт операции разного уровня сложности, транзакции и компенсации, управление данными, параллелизм, обработку исключений и операционную семантику. В основе BPML лежит язык XML, поэтому грамматика BPML оформляется в виде XML-схемы, что обеспечивает постоянство определений и их обмен между гетерогенными системами и инструментами моделирования.

BPML может применяться для более детального определения процессов. Он преобразует бизнес-операции в сообщения, которыми обмениваются процессы.

WorkFlow-процесс в BPML определяется при помощи элементов: *деятельность (Activity)*, *сигнал (Signal)*, *исключение (Exception)*, *контекст (Context)*, *свойство (Property)*.

Деятельность (Activity) является основным элементом бизнес-процесса. Элементы Activity могут соединяться последовательно (простые) или вкладываться один в другой (сложные). Разные типы деятельности выполняют различные функции.Также существуют типы деятельности, которые запускают дочерние процессы (как с ожиданием их окончания, так и без), организуют задержки выполнения процесса и т.д. Кроме того, есть несколько типов деятельности, реализующих разного вида циклы.

Синтаксис базового типа `bpml:activity` имеет вид:

```
<{activity type}
  name = NCName
  {other attributes}>
  Content: (documentation?, {other element}*)
</{activity type}>
```

BPML спецификация определяет следующие виды деятельности

— простые:

  - Action — выполняет или запускает операции обмена входящими и исходящими сообщениями;
  - Assign — присваивает новое значение свойству;
  - Call — запускает процесс и ждёт его завершения;
  - Compensate — вызывает компенсации указанных процессов;
  - Delay — выражает течение времени;
  - Empty — ничего не делает;
  - Fault — выдаёт сообщение об ошибке в текущем контексте;
  - Raise — активизирует сигнал;
  - Spawn — запускает процесс, не дожидаясь его завершения;
  - Synch — синхронизирует по сигналу;
- сложные:
  - All — выполняет указанные операции параллельно;
  - Choice — выполняет операции одного из составных комплектов, выбранного в соответствии с возникшим событием;
  - Foreach — однократно выполняет операции для каждого пункта из списка;
  - Sequence — выполняет операции последовательно;
  - Switch — выполняет операции одного из составных комплектов, выбранного на основе истинного значения условия;
  - Until — выполняет операции один или более раз на основе истинного значения условия;
  - While — не выполняет операции или выполняет их один или более раз на основе истинного значения условия.

Для деятельности определены следующие состояния:

— ready — деятельность находится в состоянии готовности;
— active — деятельность находится в активном состоянии (работает);
— completing — деятельность совершает действия для завершения выполненных работ;
— completed — деятельность полностью завершила выполняемые действия;
— aborting — деятельность совершает действия для прерывания выполняемых работ;
— aborted — деятельность полностью завершила действия для прерывания выполняемых работ.

Синтаксис для определения элемента `Action` имеет вид:

```
<action
  name = NCName
  portType = QName
  operation = NCName
  correlate = list of QName
  locate = QName
  role = QName
  {extension attribute}>
  Content: (documentation?, (input | output)*, {any activity}*)
</action>
```

Синтаксис для определения входящего элемента имеет вид:

```
<input
  property = QName
  element = QName
  xpath = XPath/>
```

Синтаксис для определения исходящего элемента имеет вид:

```
<output
  type = QName
  element = QName
  xpath = XPath>
  Content: (((source | value)+ | {extension element})?)
</output>
```

Синтаксис элемента `Assign` имеет вид:

```
<assign
  name = NCName
  property = QName
  xpath = XPath
  {extension attribute}>
  Content: (documentation?, ((source | value)+ | {extension element})?)
</assign>
```

Синтаксис элемента `Source` имеет вид:

```
<source
  property = QName
  xpath = XPath>
  Content: ({extension element})?)
</source>
```

Синтаксис элемента `Call` имеет вид:

```
<call
  name = NCName
  process = QName>
  Content: (documentation?, output*, input*)
</call>
```

Синтаксис элемента `Compensate` имеет вид:

```
<compensate
  name = NCName
  process = list of QName>
  Content: (documentation?, output*)
</compensate>
```

Синтаксис элемента `Delay` имеет вид:

```
<delay
  name = NCName
  duration = QName
  instant = QName>
  Content: (documentation?)
</delay>
```

Синтаксис элемента `Empty` имеет вид:

```
<empty
  name = NCName>
  Content: (documentation?)
</empty>
```

Синтаксис элемента `Fault` имеет вид:

```
<fault
  name = NCName
  code = QName
  property = QName>
  Content: (documentation?)
</fault>
```

Синтаксис элемента `Raise` имеет вид:

```
<raise
  name = NCName
  signal = QName
  fault = boolean : true>
  Content: (documentation?, output*)
</raise>
```

Синтаксис элемента `Spawn` имеет вид:

```
<spawn
  name = NCName
  process = QName>
  Content: (documentation?, output*)
</spawn>
```

Синтаксис элемента `All` имеет вид:

```
<all
  name = NCName>
  Content: (documentation?, context?, {any activity}+)
</all>
```

Синтаксис элемента `Choice` имеет вид:

```
<choice
  name = NCName>
  Content: (documentation?, event{2,*})
</choice>

<event>
  Content: (documentation?, (action | synch | delay),
            context?, {any activity}+)
</event>
```

Синтаксис элемента `Foreach` имеет вид:

```
<foreach
  name = NCName
  select = XPath>
  Content: (documentation?, context?, {any activity}+)
</foreach>
```

Синтаксис элемента `Sequence` имеет вид:

```
<sequence
  name = NCName>
  Content: (documentation?, context?, {any activity}+)
</sequence>
```

Синтаксис элемента `Switch` имеет вид:

```
<switch
  name = NCName>
  Content: (documentation?, case+, default?)
</switch>

<case
  name = NCName>
  Content: (documentation?, condition, context?, {any activity}+)
</case>

<default
  name = NCName>
  Content: (documentation?, context?, {any activity}+)
</default>
```

Синтаксис элемента `Until` имеет вид:

```
<until
  name = NCName>
  Content: (documentation?, condition, context?, {any activity}+)
</until>
```

Синтаксис элемента `While` имеет вид:

```
<while
  name = NCName>
  Content: (documentation?, condition, context?, {any activity}+)
</while>
```

Сигналы (Signals) используются для синхронизации точек управления, находящихся в элементах Activities одного уровня вложенности.

Синтаксис для определения сигнала имеет вид:

```
  <signal
    name = NCName
    type = QName
    element = QName
    multi = boolean : false>
    Content: (documentation?, (value | source)?)
  </signal>
  <source
    property = QName>
    Content: (condition?)
  </source>
```

Двумя важными компонентами модели BPML являются *системные ошибки (Faults)* и *планировщики (Schedules)*.

Системные ошибки (как и исключительные события) приводят к завершению выполнения текущего задания, в результате чего управление передаётся специальным обработчикам (системным или определённым в контексте).

Синтаксис для определения системных ошибок имеет вид:

```
<faults>
  Content: ((case+, default?) | default)
</faults>

<case
  name = NCName
  code = list of QName>
  Content: (documentation?, {any activity}+)
</case>

<default
  name = NCName>
  Content: (documentation?, {any activity}+)
</default>
```

При этом элемент `case` используется, если заданы более одного кода неисправности, а элемент `default` используется, если не указаны специальные ошибки кодов.

Планировщики выполняют управляющие функции: в их задачу входит генерация специальных сообщений в определённые моменты времени, что позволяет осуществлять планирование запуска процессов.

Синтаксис для определения планировщика имеет вид:

```
<schedule
  name = NCName
  process = QName
  code = QName
  duration = QName
  instant = QName
  repeat = QName>
  Content: (documentation?, {extension element}?)
</schedule>
```

Исключение (Exception) соответствует возникновению нештатной ситуации, когда оказывается, что выполнять некоторый участок бизнес-процесса уже не требуется[1].

[1]Например, во время выполнения бизнес-процесса оформления туристической поездки клиент позвонил в туристическую компанию и сообщил, что он отказывается от поездки.

Синтаксис для определения исключения имеет вид:

```
<exception
  name = NCName>
  Content: (documentation?, event, context?, {any activity}+)
</exception>
```

Компенсация (Compensation) соответствует необходимым действиям по корректному завершению ситуации, возникшей в связи с Exception[1].

Синтаксис для определения компенсации имеет вид:

```
<compensation
  name = NCName
  duration = QName
  instant = QName>
 Content: (documentation?, (event | parameters?), context?, {any activity}+)
</compensation>
```

Контекст (Context) содержит относящиеся к процессу переменные, локальные определения процессов, сигналов и т.д., служит для синхронизации и передачи информации между узлами.

Синтаксис для определения контекста имеет вид:

```
<context
  atomic = boolean : false>
  Content: ((exception | process | property | schedule | signal)*, faults?)
</context>
```

Переменные определяются набором *свойств (Property)* и могут быть локальными или глобальными по отношению к данному контексту.

Синтаксис для задания свойств имеет вид:

```
<property
  name = NCName
  type = QName
```

[1]Если до отказа клиента от поездки для клиента были забронированы билеты на самолёт и номер в гостинице, то задачей элемента Compensation будет отменить бронирование.

```
  element = QName
  fixed = boolean>
  Content: (documentation?, value?)
</property>
```

Следует отметить, что синтаксис конструкций для взаимодействия бизнес-процесса с внешними приложениями и исполнители не определены в языке BPML — все это перенесено на технологию Web-сервисов.

9.3. Язык реализации бизнес-процессов BPEL

Язык реализации бизнес-процессов (Business Process Execution Language, BPEL) используются для реализации модели нотации BPMN.

Язык BPEL стал результатом слияния WorkFlow-языков WSFL и XLANG, основанных на разных моделях: WSFL базируется на теории графов, XLANG — на иерархии тегов XML. BPEL позволяет определить последовательность выполнения функционала Web-сервисов в ходе различных потоков операций (транзакций), но при этом не учитывает, как реализованы Web-сервисы, а лишь координирует их работу в ходе бизнес-процесса.

BPEL позволяет применять условные ветвления, организовывать потоки параллельных вычислений, описывать правила соединения потоков, обмениваться данными между потоками, применять синхронные и асинхронные режимы взаимодействия, обрабатывать исключительные ситуации и т.п.

Создаваемые с помощью BPEL приложения относятся к категории *«процессно-ориентированных» (Process-Based Applications)*, состоящих из двух отдельных слоев исполнения. Верхний слой описывает бизнес-логику процесса, представленную на языке BPEL, нижний слой выполняет все функциональные операции с помощью различных Web-сервисов.

BPEL определяет два вида процессов — *абстрактный* и *исполняемый*. Абстрактный процесс определяет протокол обмена сообщениями между различными участниками без учёта алгоритма их внутреннего поведения. Ис-

полняемый процесс содержит в себе алгоритмы, определяющие порядок выполнения *действий (Activities)*, назначение исполнителей, обмен сообщениями, правила обработки исключений и т.д.

Действия (Activities) в BPEL делятся на *примитивные* и *структурные*.

К примитивным Activities относятся:

— Receive — ожидает сообщения внешнего источника;
— Reply — отвечает внешнему источнику;
— Invoke — запускает операцию какого-либо Web-сервиса;
— Wait — ждёт в течение определённого периода времени;
— Assign — копирует значение одной переменной в другую;
— Throw — отбрасывает исключение в случае ошибки;
— Terminate — принудительно завершает выполнение службы;
— Empty — не выполняет никаких действий.

К структурным Activities относятся:

— Sequence — соответствует последовательному выполнению действий, содержащихся внутри элемента;
— Switch — условная передача управления (соответствует оператору Switch языков программирования C++, Java и т. д.);
— While — организует цикл типа «While»;
— Pick — запускает обработку событий и исключительных ситуаций;
— Flow — соответствует параллельному выполнению действий, содержащихся внутри элемента;
— Scope — группирует узлы для программы-обработчика ошибок.

Кроме того, в языке присутствует понятие *связь (Link)*, которая, как правило, применяется к элементам Activities, находящимся внутри параллельного блока, накладывая ограничения на порядок их выполнения.

Переменные в BPEL описываются при помощи тега «variables». Для задания исполнителя используется тег «partnerLink».

В приложении Б.2 рассмотрен пример реализации диаграммы нотации BPMN на языке BPEL [14].

Глава 10. Другие формальные языки моделирования бизнес-процессов

10.1. Методология ARIS

Методология построения интегрированных информационных систем (Architecture of Integrated Information Systems, ARIS) предполагает определённый подход к формализации информации о деятельности организации и представление её в виде графических моделей, удобном для понимания и анализа. Модели, создаваемые по методологии ARIS, отражают существующую ситуацию с той или иной степенью приближённости. Степень детализации описания зависит от целей проекта, в рамках которого проводится моделирование. Модели ARIS могут быть использованы для анализа и выработки различного рода решений по реорганизации деятельности предприятия, в том числе по внедрению информационной системы управления, разработке систем менеджмента качества.

Методология ARIS реализует принципы структурного анализа и позволяет определить и отразить в моделях основные компоненты организации, протекающие процессы, производимую и потребляемую продукцию, используемую информацию, а также выявить взаимосвязи между ними. Создаваемые модели представляют собой документированную совокупность знаний о системе управления, включая организационную структуру, протекающие процессы, взаимодействия между организацией и субъектами рынка, состав и структуру документов, последовательность шагов процессов, должностные инструкции отделов и их сотрудников. В отличие от других подходов, методология ARIS предполагает хранение всей информации в едином репозитории, что обеспечивает целостность и непротиворечивость процесса моделирования и анализа, а также позволяет проводить верификацию моделей. Любая

организация рассматривается как единая система, описание которой предусматривает четыре основные группы моделей:

— модели организационной структуры;
— модели данных (потоки и структура);
— модели функций (функциональные иерархии);
— модели контроля и управления (сводные модели бизнес-процессов).

При этом каждая из этих точек зрения разделяется ещё на три подуровня: *описание требований*, *описание спецификации*, *описание внедрения*. Для описания бизнес-процессов предлагается использовать около 80 типов моделей, каждая из которых принадлежит тому или иному аспекту.

Архитектура ARIS включает большое количество типов моделей, использующих различные графические объекты для построения разносторонних моделей организации. Но на практике применяется ограниченное число нотаций архитектуры ARIS, например:

— метод описания процессов (Event–Driven Process Chain, EPC);
— модель сущность–связь для описания структуры данных (Entity Relationship Model, ERM);
— язык моделирования UML.

К сожалению, нотация ARIS поприетарная и её не поддерживает ни одно инструментальное средство, кроме IDS Scheer ARIS.

Нотация ARIS *Extended Event Driven Proce-Chain (eEPC)* представляет собой расширенную нотацию описания цепочки процесса, управляемого событиями. По сути, она является расширением методологии IDEF3 за счёт использования такого понятия, как *событие (Event)*. Кроме нотации eEPC, ARIS предоставляет аналитику и другие средства описания процессов организации.

Основные объекты, используемые в рамках нотации:

— *функция (Function)* служит для описания функций (процедур, работ), выполняемых подразделениями/сотрудниками предприятия;
— *событие (Event)* служит для описания реальных состояний системы, влияющих и управляющих выполнением функций;

- *организационная единица (Organizational Unit)* обозначает различные организационные звенья компании (например, управление или отдел);
- *документ (Document)* представляет собой реальные носители информации, например, бумажную документацию, в рамках технологии выполнения функции;
- *кластер информации (Cluster)* характеризует данные как набор сущностей и связей между ними, используется для создания моделей данных;
- *стрелка связи между объектами* описывает тип отношений между другими объектами, например, активацию выполнения функции некоторым событием;
- *оператор логическое «И» (AND)* определяет связи между событиями и функциями в рамках процесса;
- *оператор логическое «ИЛИ» (OR)* определяет связи между событиями и функциями в рамках процесса;
- *оператор логическое исключающее «ИЛИ» (XOR)* определяет связи между событиями и функциями в рамках процесса.

Используемые при построении модели символы логики позволяют отразить ветвление и слияние бизнес-процессов.

Реальная длительность выполнения процедур в eEPC не может быть отражена визуально. Для получения информации о реальной длительности процессов необходимо использовать другие инструменты описания, например *диаграммы Ганта*.

Нотация eEPC построена на определённых семантических правилах описания:

- каждая функция должна быть инициирована событием и должна завершаться событием;
- в каждую функцию может входить не более одной стрелки, «запускающей» выполнение функции, и выходить не более одной стрелки, описывающей завершение выполнения функции.

Заключение

Данное пособие не ставило перед собой цель углублённого изучения какой-либо нотации бизнес-проектирования. Цель — дать представление читателю о существующих методологиях и нотациях в области бизнес-проектирования, дабы в дальнейшем он мог применить любую из них в зависимости от своих потребностей.

В книге представлены три основных подхода к проектированию бизнес-процессов: IDEF, UML, BPMN. IDEF является, по сути, прародителем всех существующих нотаций, объединяющим множество методик, применяемых при проектировании бизнес-процессов. Разработка модели не требует от аналитика дополнительных технических знаний, лишь чёткого представления о деятельности анализируемой системы. UML пытается ввести единый язык для описания и построения бизнес-моделей, но при этом представляет из себя достаточно громоздкий и непривычный для бизнес-аналитиков инструмент. Поэтому он закономерно нашёл своё применение в среде проектирования программных систем, поскольку разработчики программ владеют объектным подходом изначально. BPMN же представляется достаточно удачной попыткой замены нотации IDEF. При этом в BPMN используются привычные для бизнес-аналитиков методологии проектирования бизнес-процессов, и в то же время вносит достаточно существенные изменения в нотацию. Следует отметить, что BPMN не отделим от интегрированной среды разработки в отличие от IDEF, который даже без использования интегрированных средств проектирования не теряет своей мощности.

Подходы к бизнес-проектированию, рассмотренные в данной книге, в той или иной степени являются базовыми для существующих инструментов организационного проектирования. Выбор методологии и инструмента зависит лишь от постановки задачи и преследуемых целей.

Приложение А. Стандартные элементы UML

А.1. Стандартные стереотипы UML

В табл. А.1 приведены стереотипы, которые определены в UML как стандартные элементы [13, 15].

Таблица А.1

Основные стереотипы UML

Стереотип	**Элемент**	**Назначение стереотипа**
actor	Класс	Определяет связанное множество ролей, которые играет пользователь при взаимодействии с объектом системы.
access	Зависимость	Сообщает, что открытое содержание целевого пакета доступно в пространстве имён исходного пакета.
association	Концевая точка связи	Указывает, что соответствующий объект видим ассоциацией.
become	Сообщение	Целевой объект совпадает с исходным, но в более поздний момент времени (при этом, возможно, у него будут другие значения, состояния или роли).
bind	Зависимость	Исходный класс инстанцирует целевой шаблон с данными фактическими параметрами.
call	Зависимость	Исходная операция вызывает целевую.
copy	Сообщение	Точная, но независимая копия исходного объекта.

Таблица А.1

Основные стереотипы UML (продолжение)

Стереотип	**Элемент**	**Назначение стереотипа**
create	Событие, сообщение	Целевой объект создан в результате события или сообщения.
derive	Зависимость	Исходный объект может быть вычислен по целевому.
destroy	Событие, сообщение	Целевой объект уничтожен в результате события или сообщения.
document	Компонент	Компонент представляет документ.
enumeration	Класс	Определяет перечисляемый тип, включая его возможные значения как набор идентификаторов.
exception	Класс	Определяет событие, которое может быть вызвано или перехвачено операцией.
executable	Компонент	Описывает компонент, который может быть выполнен в узле.
extend	Зависимость	Целевой вариант использования расширяет поведение исходного варианта использования в данной точке расширения.
facade	Пакет	Пакет, который является лишь представлением другого пакета.
file	Компонент	Компонент, который представляет документ, содержащий исходный код или данные.

Таблица А.1

Основные стереотипы UML (продолжение)

Стереотип	**Элемент**	**Назначение стереотипа**
framework	Пакет	Пакет, состоящий в основном из образцов.
friend	Зависимость	Исходный класс имеет специальные права видимости.
global	Концевая точка связи	Соответствующий объект видим, поскольку принадлежит объемлющей области действия.
import	Зависимость	Открытое содержание целевого пакета становится частью плоского пространства имён исходного пакета, как если бы оно было объявлено непосредственно в нём.
implementation	Обобщение	Потомок наследует реализацию родителя, но не открывает и не поддерживает его интерфейсов, вследствие чего не может быть подставлен вместо родителя.
implementationClass	Класс	Реализация класса на некотором языке программирования.
include	Зависимость	Исходный вариант использования явно включает поведение другого варианта использования в точке, определяемой исходным вариантом использования.

Таблица А.1

Основные стереотипы UML (продолжение)

Стереотип	**Элемент**	**Назначение стереотипа**
instanceOf	Зависимость	Исходный объект является экземпляром целевого классификатора.
instantiate	Зависимость	Операции над исходным классом создают экземпляры целевого класса.
interface	Класс	Описывает множество операций, определяющих, что может делать класс или компонент.
invariant	Ограничение	Ограничение, которое всегда должно выполняться для ассоциированного элемента.
library	Компонент	Статическая или динамическая объектная библиотека.
local	Концевая точка связи	Соответствующий объект видим, так как находится в локальной области действия.
metaclass	Классификатор	Классификатор, все объекты которого являются классами.
model	Пакет	Описывает семантически замкнутую абстракцию системы.
parameter	Концевая точка связи	Соответствующий объект видим, так как является параметром.
postcondition	Ограничение	Ограничение, которое должно выполняться после выполнения операции.

Таблица А.1

Основные стереотипы UML (продолжение)

Стереотип	**Элемент**	**Назначение стереотипа**
powertype	Класс	Классификатор, все объекты которого являются потомками данного родителя.
precondition	Ограничение	Ограничение, которое должно выполняться перед выполнением операции.
process	Класс	Классификатор, экземпляр которого представляет ресурсоёмкий поток управления.
refine	Зависимость	Говорит, что исходный объект является более детальной абстракцией, чем целевой объект.
requirement	Комментарий	Описывает желаемое свойство или поведение системы.
responsibility	Комментарий	Описывает контракт или обязательство класса.
send	Зависимость	Исходная операция посылает целевое событие.
signal	Класс	Асинхронный сигнал, который передаётся одним экземпляром другому.
stereotype	Класс	Стереотип, который может быть применён к другим элементам.
stub	Пакет	Пакет выступает в роли заместителя для открытого содержимого другого пакета.

Таблица А.1

Основные стереотипы UML (продолжение)

Стереотип	**Элемент**	**Назначение стереотипа**
subsystem	Пакет	Описывает группирование элементов, ряд которых составляет спецификацию поведения других элементов.
system	Пакет	Описывает пакет, представляющий всю моделируемую систему.
table	Компонент	Компонент, представляющий таблицу базы данных.
thread	Класс	Классификатор, экземпляр которого представляет облегчённый поток управления.
trace	Зависимость	Исторический предок исходного элемента.
type	Класс	Абстрактный класс, который используется только для спецификации структуры и поведения (но не реализации) множества объектов.
use	Зависимость	Семантика исходного элемента зависит от семантики открытого содержания целевого элемента.
utility	Класс	Определяет класс, для которого область действия всех атрибутов и операций — класс.

А.2. Стандартные помеченные значения UML

В табл. А.2 приведены помеченные значения, которые определены в UML как стандартные элементы [13, 15].

Таблица А.2

Основные помеченные значения UML

Помеченное значение	**Элемент**	**Назначение помеченного значения**
documentation	Все элементы	Содержит комментарий, описание или пояснение к элементу.
location	Большинство элементов	Определяет узел или компонент, которому принадлежит элемент.
persistence	Класс, ассоциация, атрибут	Определяет, сохраняется ли состояние экземпляра после завершения создавшего его процесса.
semantics	Класс, операция	Описывает назначение класса или операции.

А.3. Стандартные ограничения UML

В табл. А.2 приведены ограничения, которые определены в UML как стандартные элементы [13, 15].

Таблица А.3

Основные ограничения UML

Ограничение	**Элемент**	**Назначение ограничения**
complete	Обобщение	В модели специфицированы все потомки в данном обобщении (хотя некоторые могут быть скрыты на диаграммах), и дополнительных потомков определять не разрешается.
destroyed	Экземпляр, связь	Экземпляр или связь уничтожаются до завершения выполнения объемлющего взаимодействия.
disjoint	Обобщение	Объекты данного родителя могут иметь не более одного заданного потомка в качестве типа.
implicit	Ассоциация	Отношение является не явно выраженным, а концептуальным.
incomplete	Обобщение	Специфицированы не все потомки в обобщении (учитывая и скрытые) и разрешается определять дополнительные потомки.
new	Экземпляр, связь	Экземпляр или связь создаются в процессе выполнения объемлющего взаимодействия.

Таблица А.3

Основные ограничения UML (окончание)

Ограничение	**Элемент**	**Назначение ограничения**
or	Ассоциация	Из множества ассоциаций ровно одна является явно выраженной для каждого ассоциированного объекта.
overlapping	Обобщение	Объекты данного родителя могут иметь более одного заданного потомка в качестве типа.
transient	Экземпляр, связь	Экземпляр или связь создаются в процессе выполнения объемлющего взаимодействия, но уничтожаются до его завершения.

Приложение Б. Реализация BPMN-диаграмм

Б.1. Пример реализации диаграммы нотации BPMN на языке BPML

В следующем примере показано использование исключений, обработчиков системных ошибок и компенсаций (рис. Б.1).

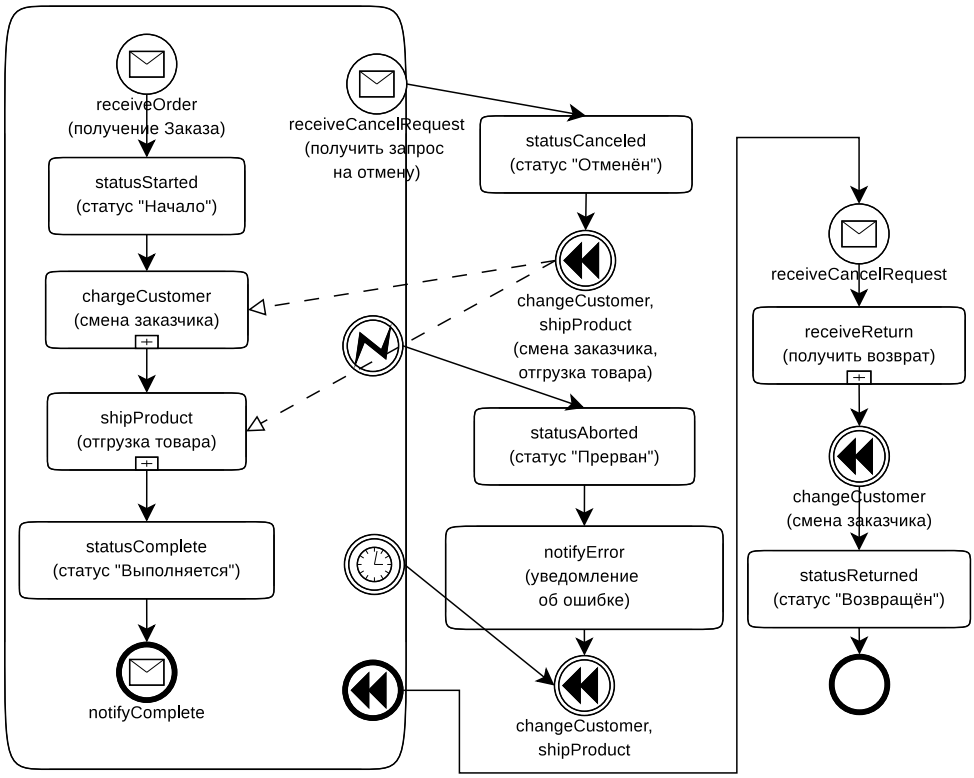


Рис. Б.1. Пример использования исключений, обработчиков системных ошибок и компенсаций

Процесс начинается с запроса на получение заказа, определения дополнительных данных о заказе и времени его выполнения. Операции синхронизируются и возвращают идентификатор заказа, используемый для корреляции последующих сообщений, отправленных или полученных в результате этого

процесса.

Процесс выполняется в два этапа, ссылаясь на процессы `chargeCustomer` (смена заказчика) и `shipProduct` (отгрузка товара), после чего посылает уведомление о выполнении. Процесс может быть прерван в любой момент после отправки запроса на отмену, что приведёт к запуску `cancelRequest` (запроса на отмену). Если процесс не завершится в течение оговорённого срока, то он будет прерван.

Если происходит ошибка, препятствующая завершению процесса, то процесс направляет соответствующее уведомляющее сообщение, прежде чем прервать свою деятельность. Уведомление не требуется, если процесс прервался по запросу на отмену или по истечении заданного времени, поскольку пользователь знает об этих условиях.

Поскольку в процессе завершения выполнения действий `chargeCustomer` (смена заказчика) и/или `shipProduct` (отгрузка товара) может возникнуть ошибка или исключительная ситуация, то используются компенсации процесса (одна выдаёт возврат, а другая — отменяет отгрузку товара). Как только этот процесс завершён, можно компенсировать этот процесс, направив запрос на отмену. Эта же операция встречается в другом контексте, когда заказ уже отправлен, а процесс ждёт соответствующего уведомления.

Далее приводится реализация описанного выше процесса на языке BPML.

```
<identity name="order" property="tns:orderID"/>

<property name="orderID" type="tns:orderID"/>
<process name="twoStepOrder" identity="tns:order">

<event activity="receiveOrder"/>

<context>
  <property ref="tns:orderID" fixed="true"/>
  <property name="details" element="tns:orderDetailsType"/>
  <property name="tns:timeLimit" type="xsd:duration"/>
  <schedule name="timeToComplete" code="tns:timeout"
     duration="tns:timeLimit"/>
```

```
  <exception name="cancelRequest">
    <event activity="receiveCancelRequest"/>
    <action name="receiveCancelRequest"
            portType="orderService" operation="cancelRequest"
            correlate="tns:orderID">
      <input element="tns:orderID" property="tns:orderID"/>
    </action>
    <assign name="statusCanceled" property="status">
      <value>canceled</value>
    </assign>
    <compensate process="tns:activityA tns:activityB"/>
  </exception>

  <faults>

    <case code="tns:timeout">
      <assign property="status">
        <value>timeout</value>
      </assign>
      <compensate process="tns:activityA tns:activityB"/>
    </case>

    <default>
      <assign name="statusAborted" property="status">
        <value>aborted</value>
      </assign>
      <action name="notifyError" portType="orderService"
          operation="notifyError">
        <output element="tns:orderID">
          <source property="tns:orderID"/>
        </output>
        <output element="tns:reason">
          <source property="inst:fault"/>
        </output>
      </action>
      <compensate process="tns:activityA tns:activityB"/>
    </default>
  </faults>
</context>

<action name="receiveOrder" portType="orderService" operation="order">
```

```
  <input element="tns:details" property="tns:orderDetails"/>
  <input element="tns:timeToComplete" property="tns:timeLimit"/>
  <output element="tns:orderID">
    <source property="tns:orderID"/>
  </output>
  <assign property="tns:orderID" xpath="func:newIdentifier('tns:orderID')"/>
</action>

<assign name="statusStarted" property="status">
  <value>started</value>
</assign>

<call process="tns:chargeCustomer">
  <output parameter="tns:details">
    <source property="tns:details"/>
  </output>
</call>

<call process="tns:shipProduct">
  <output parameter="tns:details">
    <source property="tns:details"/>
  </output>
</call>

<assign name="statusComplete" property="status">
  <value>complete</value>
</assign>

<action name="notifyComplete"
     portType="orderService" operation="notifyCompletion">
  <output element="tns:orderID">
    <source property="tns:orderID"/>
  </output>
</action>

<compensation name="cancelRequest">
  <event activity="receiveCancelRequest"/>
  <action name="receiveCancelRequest"
          portType="orderService" operation="cancelRequest"
          correlate="tns:orderID">
    <input element="tns:orderID" property="tns:orderID"/>
  </action>
```

```
  <call process=”tns:receiveReturn”>
    <output parameter=”tns:details”>
      <source property=”tns:details”/>
    </output>
  </call>
  <compensate process=”tns:chargeCustomer”/>
  <assign name=”statusReturned” property=”status”>
      <value>returned</value>
  </assign>
</compensation>
</process>
```

Б.2. Пример реализации диаграммы нотации BPMN на языке BPEL

Рассмотрим пример «Бронирование путёвки» (рис. Б.2) [14].

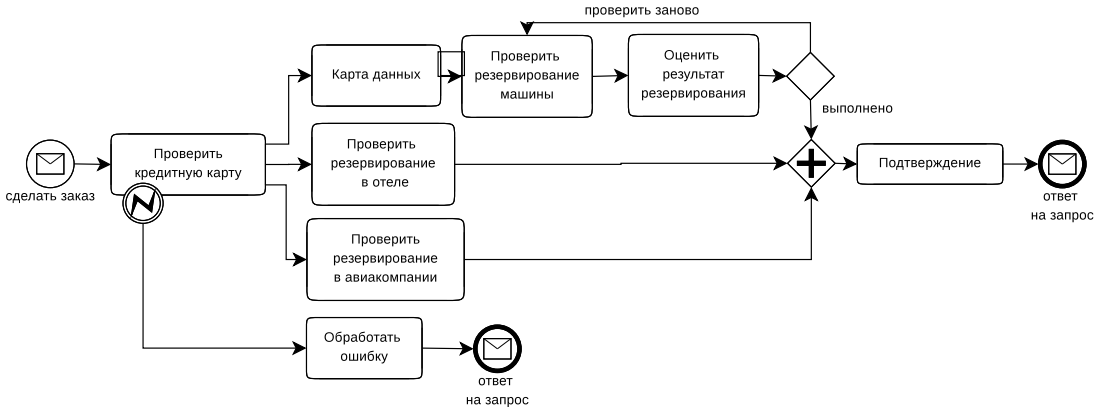


Рис. Б.2. Пример BPMN-процесса «Бронирование билетов»

Процесс начинается с получения запроса о предоставлении бронирования путёвки. После проверки кредитной карты осуществляется бронирование билета на самолёт, гостиницы и машины. Оформление бронирования автомобиля может быть выполнено не с первой попытки. После подтверждения всех заказов клиенту отправляется ответ на запрос.

Анализируя эту схему, нужно обратить внимание на то, что она показыва-

ет общую логику процесса, передачу запросов и ответов, никак не детализируя функционирование самих удалённых Web-сервисов. На языке BPEL нужно только описать последовательность обращений и способы обработки получаемой информации. Для написания соответствующей программы можно использовать один из многих BPEL-инструментов, которые на основе такой визуальной диаграммы автоматически сгенерируют код на языке BPEL.

Для создания BPEL-файла необходимо определить параметры процесса.

В параметре `partnerLink` определены элементы, предшествующие определению процесса, но информация об этих элементах находится из свойств объекта «Задача» BPMN-процесса. Задачи процесса, относящиеся к типу обслуживания и реализованные в виде web-сервиса, определяют участников web-служб. Участники и их свойства будут указывать на элементы `partnerLink`.

```
<partnerLinks>
 <partnerLink
    myRole="travelProcessRole"
    name="ProcessStarter"
    partnerLinkType="wsdl5:travelProcess"/>
 <partnerLink
    name="HotelReservationService"
    partnerLinkType="wsdl5:HotelReservationPartnerPLT"
    partnerRole="HotelReservationRole"/>
</partnerLinks>
```

Элементы `variable` также предшествуют определению процесса. Они будут созданы на основе свойств, ассоциированных с процессом BPMN-диаграммы. Элементы `variable` будут определены в документе BPEL и связаны с элементами `message`.

```
<variables>
  <variable
    messageType="wsdl0:input"
    name="input"/>
  <variable
    messageType="wsdl4:doCreditCardCheckingRequest"
    name="checkCreditCardRequest"/>
```

```
  <variable
    messageType="wsdl4:doCreditCardCheckingResponse"
    name="checkCreditCardResponse"/>
  <variable messageType="wsdl4:Exception"
    name="creditCardFault"/>
  <variable
    messageType="wsdl1:doCarReservationRequest"
    name="carReservationRequest"/>
</variables>

<message name="input">
   <part name="airline" type="xsd:string"/>
   <part name="arrival" type="xsd:string"/>
   <part name="departure" type="xsd:string"/>
</message>
<message name="doFlightReservationRequest">
   <part name="airline" type="xsd:string"/>
   <part name="arrival" type="xsd:string"/>
   <part name="departure" type="xsd:string"/>
</message>
```

Логика самого бизнес-процесса, который состоит из операций, называемых действиями, заключена внутри тегов `<flow>`. В начале этого блока находятся конструкции, называемые `links`, которые указывают направления связей при выполнении последующих операций. При этом идентификаторы `name` элементов `link` будут сгенерированы автоматически.

```
<flow name="Flow" wpc:id="1"/>
  <links>
    <link name="link1"/>
    <link name="link2"/>
    <link name="link3"/>
    <link name="link4"/>
    <link name="link5"/>
    <link name="link6"/>
    <link name="link7"/>
    <link name="link8"/>
    <link name="link9"/>
    <link name="link10"/>
    <link name="link11"/>
```

```
    <link name="link12"/>
 </links>
</flow>
```

Сообщение «Сделать заказ» в BPEL будет выглядеть следующим образом:

```
<receive
    createInstance="yes"
    operation="book"
    name="Receive"
    wpc:displayName="Receive"
    portType="wsdl0:travelPort"
    variable="input" wpc:id="2">
  <source linkName="link1" />
</receive>
```

Операция «Проверить кредитную карту» в BPEL будет выглядеть следующим образом:

```
<assign name="DataMap1" wpc:displayName="DataMap1" wpc:id="20">
  <target linkName="link1"/>
  <source linkName="link2"/>
  <copy>
      <from part="cardNumber" variable="input"/>
      <to part="cardNumber" variable="checkCreditCardRequest"/>
  </copy>
  <copy>
      <from part="cardType" variable="input"/>
      <to part="cardType" variable="checkCreditCardRequest"/>
  </copy>
</assign>
```

Каждая операция `invoke`, представленная на диаграмме прямоугольным блоком, описывается простой BPEL-структурой, которая включает элементы, соответствующие таким действиям, как отправка запроса, ожидание, получение ответа и т.д. Действие `invoke` может применяться в синхронном режиме для выполнения операции «отправить–принять».

```
<invoke
   inputVariable=”checkCreditCardRequest”
   name=”checkCreditCard”
   operation=”doCreditCardChecking”
   outputVariable=”checkCreditCardResponse”
   partnerLink=”CreditCardCheckingService”
   portType=”wsdl4:CreditCardCheckingServiceImpl”
   wpc:displayName=”Check Credit Card” wpc:id=”5”>
  <target linkName=”link2”/>
  <source linkName=”link3”/>
  <source linkName=”link6”/>
  <source linkName=”link9”/>
</invoke>
```

Операция «Проверить бронь в авиакомпании» в BPEL будет выглядеть следующим образом:

```
<assign name=”DataMap2” wpc:displayName=”DataMap2” wpc:id=”21”>
  <target linkName=”link3”/>
  <source linkName=”link4”/>
  <copy>
     <from part=”airline” variable=”input”/>
     <to part=”airline” variable=”flightReservationRequest”/>
  </copy>
</assign>
<invoke
    inputVariable=”flightReservationRequest”
    name=”checkFlightReservation”
    operation=”doFlightReservation”
    outputVariable=”flightReservationResponse”
    partnerLink=”FlightReservationService”
    portType=”wsdl3:FlightReservationServiceImpl”
    wpc:displayName=”Check Flight Reservation”
    wpc:id=”10”>
  <target linkName=”link4”/>
  <source linkName=”link5”/>
</invoke>
```

Операция «Проверить бронь в отеле» в BPEL будет выглядеть следующим образом:

```
<assign name="DataMap3" wpc:displayName="DataMap3" wpc:id="22">
  <target linkName="link6"/>
  <source linkName="link7"/>
  <copy>
     <from part="hotelCompany" variable="input"/>
     <to part="name" variable="hotelReservationRequest"/>
  </copy>
</assign>
<invoke
    inputVariable="hotelReservationRequest"
    name="checkHotelReservation"
    operation="doHotelReservation"
    outputVariable="hotelReservationResponse"
    partnerLink="HotelReservationService"
    portType="wsdl2:HotelReservationServiceImpl"
    wpc:displayName="Check Hotel Reservation"
    wpc:id="9">
 <target linkName="link7"/>
 <source linkName="link8"/>
</invoke>
```

Операция «Карта данных» в BPEL будет выглядеть следующим образом:

```
<assign name="DataMap4" wpc:displayName="Data Map" wpc:id="23">
 <target linkName="link9"/>
 <source linkName="link10"/>
 <copy>
    <from part="carCompany" variable="input"/>
    <to part="company" variable="carReservationRequest"/>
 </copy>
</assign>
```

Повторная операция «Проверка брони на машину» в BPEL будет выглядеть следующим образом:

```
<while condition="DefinedByJavaCode" name="While" wpc:id="11">
 <wpc:condition>
   <wpc:javaCode><![CDATA[
     boolean condition = false;
     try {
       if (getCarReservationResponse().getBooleanPart("result")) {
```

```
            condition = false;
        } else {
            condition = true;
          }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return condition;
   ]]>
   </wpc:javaCode>
   </wpc:condition>
   <target linkName="link10"/>
   <source linkName="link11"/>
   <flow wpc:id="16">
   <links>
   <link name="link13"/>
   </links>
 </flow>
</while>

<invoke
   inputVariable="carReservationRequest"
   name="checkCarReservation"
   operation="doCarReservation"
   outputVariable="carReservationResponse"
   partnerLink="CarReservationService"
   portType="wsdl1:CarReservationServiceImpl"
   wpc:displayName="Check Car Reservation"
   wpc:id="13">
 <source linkName="link13"/>
</invoke>
```

Операция «Оценить результат бронирования» в BPEL будет выглядеть следующим образом:

```
<invoke name="evaluateReservationRequest"
    operation="null" partnerLink="null" portType="wpc:null"
    wpc:displayName="Evaluate Reservation Request" wpc:id="14">
    <wpc:script>
        <wpc:javaCode><![CDATA[
            <!-- Java Code Inserted here -->
```

```
                                    ]]>
        </wpc:javaCode>
    </wpc:script>
    <target linkName="link13"/>
</invoke>
```

Операция «Подтверждение заказа» в BPEL будет выглядеть следующим образом:

```
<invoke name="Confirmation" operation="null"
    partnerLink="null" portType="wpc:null"
    wpc:displayName="Confirmation" wpc:id="12">
  <wpc:script>
      <wpc:javaCode><![CDATA[
          <!-- Java Code Inserted here -->
                                  ]]>
  </wpc:javaCode>
  </wpc:script>
  <target linkName="link5"/>
  <target linkName="link8"/>
  <target linkName="link11"/>
  <source linkName="link12"/>
</invoke>
```

Отправка «Ответа на запрос» в BPEL будет выглядеть следующим образом:

```
<reply
    name="Reply" operation="book"
    wpc:displayName="Reply"
    partnerLink="ProcessStarter"
    portType="wsdl0:travelPort"
    variable="output" wpc:id="3">
  <target linkName="Link12"/>
</reply>
```

Операция «Обработать ошибку» в BPEL будет выглядеть следующим образом:

```
<scope wpc:id="15">
  <faultHandlers>
     <catch
        faultName="wsdl4:Exception"
        faultVariable="creditCardFault">
     <flow wpc:id="17">
        <links>
           <link name="Link14"/>
           </links>
           <invoke name="HandleFault" operation="null"
              wpc:displayName="handleFault" partnerLink="null"
              portType="wpc:null" wpc:id="18">
              <wpc:script>
              <wpc:javaCode><![CDATA[
                 <!-- Java Code Inserted here -->
                                    ]]>
              </wpc:javaCode>
              </wpc:script>
              <source linkName="Link14"/>
           </invoke>
           <reply name="Reply" operation="book"
              wpc:displayName="Reply" partnerLink="ProcessStarter"
              portType="wsdl0:travelPort" variable="output"
              wpc:id="19">
            <target linkName="Link14"/>
           </reply>
     </flow>
     </catch>
  </faultHandlers>
 <flow wpc:id="15">
 </flow>
</scope>
```

Таким образом, этот пример показывает, как диаграмма бизнес-процесса BPMN может быть использована для представления исполняемого процесса. Для создания исполняемого процесса были проанализированы диаграммы объектов и их свойства, а затем отображены в соответствующие BPEL-элементы.

Список иллюстраций

Список таблиц

Используемая литература

1. *Вендров А. М.* Методы и средства моделирования бизнес-процессов (обзор) // JetInfo. — № 10 (137). — 2004. — http://www.jetinfo.ru/2004/10/1/article1.10.2004.html.
2. Quality management systems — Fundamentals and vocabulary, ISO 9000:2005. — 2005. — http://www.iso.org.
3. Integration Definition for Function Modeling (IDEF0). Software Standard, Modeling Techniques. — National Institute of Standards and Technology, 1993.
4. IDEF3 Process Description Capture Method Report: Technical report al-tr-1995-xxxx / R. J. Mayer, C. P. Menzel, M. K. Painter et al / Knowledge Based Systems, Inc. — 1995.
5. IDEF4 Object-Oriented Design Method Report. Version 2.0: Technical report / Knowledge Based Systems, Inc. — 1995.
6. IDEF5 Method Report: Technical report / P. C. Benjamin, C. P. Menzel, R. J. Mayer et al / Knowledge Based Systems, Inc. — 1994.
7. *Mayer R. J., Painter M. K., Lingineni M.* Toward a Method for Business Constraint Discovery (IDEF9): Technical report / Knowledge Based Systems, Inc. — 1995.
8. Information technology – Open Distributed Processing – Unified Modeling Language (UML) Version 1.4.2, ISO/IEC 19501:2005. — 2005. — http://www.iso.org.
9. Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification dtc/06-02-01. — OMG, 2006. — http://www.bpmn.org.
10. Business Process Model and Notation (BPMN) 2.0. OMG Document: BMI/2007-06-05. — 2007. — http://www.bpmn.org.
11. Методология функционального моделирования. Рекомендации по стандартизации Р 50.1.028–2001. — М.: ИПК Издательство стандартов, 2001.

12. *Леоненков А. В.* Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose. — Интернет-университет информационных технологий — ИНТУИТ.ру, БИНОМ. Лаборатория знаний, 2006. — 320 с.
13. *Буч Г., Рамбо Д., Якобсон А.* Язык UML. Руководство пользователя. 1-е издание. — Питер, 2004. — 432 с.
14. *White S. A.* Using BPMN to Model a BPEL Process // OMG/BPMI. — 2005. — http://www.bpmn.org/Documents/MappingBPMNtoBPELExample.pdf.
15. *Рамбо Д., Якобсон А., Буч Г.* UML. Специальный справочник. — Питер, 2001. — 656 с.
16. *Черемных С. В., Семёнов И. О., Ручкин В. С.* Структурный анализ систем: IDEF-технологии. — М.: Финансы и статистика, 2001.
17. *Калашян А. Н., Калянов Г. Н.* Структурные модели бинеса: DFD-технологий. — М.: Финансы и статистика, 2003. — 256 с.
18. Федеральное агенство по техническому регулированию и метрологии. — http://www.gost.ru.
19. *Верников Г.* Обзор стандарта IDEF0. — http://idefinfo.ru/content/view/12/27/.
20. *Курьян А. Г., Серенков П. С.* Использование IDEF0 для описания и классификации процессов в рамках системы качества ISO 9000. — http://idefinfo.ru/content/view/22/27/.
21. *Новиков М. В.* IDEF0 в моделировании бизнес-процессов управления. — http://idefinfo.ru/content/view/21/27/.
22. Integration Definition for Function Modeling (IDEF1X). Software Standard, Modeling Techniques. — National Institute of Standards and Technology, 1993.
23. IDEF1 Information Modeling: Technical report afwal-tr-81-4023 / Ed. by R. J. Mayer / Knowledge Based Systems, Inc. — 1992.

24. *Верников Г.* Обзор стандарта IDEF1. — http://idefinfo.ru/content/view/14/27/.
25. *Верников Г.* Обзор стандарта IDEF1x. — http://idefinfo.ru/content/view/17/27/.
26. *Верников Г.* Обзор стандарта IDEF3. — http://idefinfo.ru/content/view/18/27/.
27. International Organization for Standardization. — http://www.iso.org.
28. *Шеер А.-В.* Моделирование бизнес-процессов. Издание 2-е, переработанное и дополненное. Перевод с английского. — Москва: Весть–МетаТехнология, 2000.
29. *Боггс У., Боггс М.* UML и Rational Rose. — Лори, 2000. — 582 с.
30. *Боггс У., Боггс М.* UML и Rational Rose 2002. — Лори, 2004. — 528 с.
31. *Owen M., Raj J.* BPMN and Business Process Management. Introduction to the New Business Process Modeling Standard. — Popkin Software, 2003.
32. *White S. A.* Process Modeling Notations and Workflow Patterns // OMG/BPMI. — 2004.
33. *Войнов Н.* Практика использования BPMN. — 2007. — http://nvoynov.googlepages.com/bpmn-practice.pdf.
34. *Войнов Н.* Эффективная работа в Intalio|BPMS. — 2007. — http://nvoynov.googlepages.com/intalio-bpms_best-practices.pdf.
35. *Черняк Л.* BPM: близкие перспективы и далекие горизонты // Открытые системы. — № 11. — 2004.
36. *Волков Ю. О.* Новый взгляд на описание бизнес-процессов // PC Week/Russian Edition. — № 34. — 2005. — С. 42, 55.
37. *Волков Ю. О.* Диаграммы для описания бизнес-процессов. — 2006. — http://yurivolkov.com/articles/Diagrams_for_business_processes_ru.html.

38. *Репин В. В.* Сравнительный анализ нотаций // finexpert.ru. — 2001. — http://www.finexpert.ru,http://www.interface.ru/fset.asp?Url=/ca/an/danaris1.htm.
39. Использование механизмов интеграции приложений для автоматизации процессов управления информационными ресурсами / А. К. Нестеренко, А. А. Данилина, Т. М. Сысоев и др // Труды 8-ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» — RCDL'2006. — Суздаль, Россия: 2006.
40. *Фирсова Н. В.* Инструментальные средства моделирования бизнес-процессов и оценка их применения для целей реинжиниринга // Вестник СПбГУ. Сер. 8. — Вып. 4. — 2005. — http://vestnikmanagement.pu.ru/archive/pdf/214.pdf.

Рекомендуемая литература

1. Методология функционального моделирования. Рекомендации по стандартизации Р 50.1.028–2001. — М.: ИПК Издательство стандартов, 2001.
2. *Боггс У., Боггс М.* UML и Rational Rose. — Лори, 2000. — 582 с.
3. *Боггс У., Боггс М.* UML и Rational Rose 2002. — Лори, 2004. — 528 с.
4. *Буч Г., Рамбо Д., Якобсон А.* Язык UML. Руководство пользователя. 1-е издание. — Питер, 2004. — 432 с.
5. *Войнов Н.* Практика использования BPMN. — 2007. — `http://nvoynov.googlepages.com/bpmn-practice.pdf`.
6. *Калашян А. Н., Калянов Г. Н.* Структурные модели бинеса: DFD-технологий. — М.: Финансы и статистика, 2003. — 256 с.
7. *Леоненков А. В.* Объектно-ориентированный анализ и проектирование с использованием UML и IBM Rational Rose. — Интернет-университет информационных технологий — ИНТУИТ.ру, БИНОМ. Лаборатория знаний, 2006. — 320 с.
8. *Рамбо Д., Якобсон А., Буч Г.* UML. Специальный справочник. — Питер, 2001. — 656 с.
9. *Черемных С. В., Семёнов И. О., Ручкин В. С.* Структурный анализ систем: IDEF-технологии. — М.: Финансы и статистика, 2001.
10. *Шеер А.-В.* Моделирование бизнес-процессов. Издание 2-е, переработанное и дополненное. Перевод с английского. — Москва: Весть–МетаТехнология, 2000.

Предметный указатель