Министерство образования и науки Российской Федерации

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Н.В. Зариковская

Анализ и разработка моделей информационных процессов и структур

Учебно-методическое пособие

Томск, 2018

Зариковская Н.В. Анализ и разработка моделей информационных процессов и структур. Учебно-методическое пособие - Томск: Изд-во ТУСУР, 2018. - 169 с.

Рассмотрены вопросы применение современных языков и инструментов для моделирования предметной области автоматизации. Приведены современные парадигмы и инструменты моделирования, возможности различных инструментов по описанию предметной области автоматизации на различных этапах создания информационных систем. Особое внимание уделяется объектно-ориентированному анализу и проектированию на базе инструмента Enterprise Architect и методологии структурного анализа и проектирования на базе AllFusion Modeling Suite.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Повышение сложности разрабатываемых систем, разделение специалистов на роли, высокий процент неудачных проектов и повышение требований к формальному описанию систем привели к тому, что в индустрии разработки ПО возникла настоятельная необходимость перехода к промышленному процессу производства ПО. Под промышленным процессом производства подразумевается введение корпоративного стандарта процесса, адаптированного под потребности организации, а также внедрение в процесс разработки специализированных инструментальных средств поддержки этого процесса. В настоящее время на рынке имеется большое количество такого инструментария разной степени функциональности, стоимости и степени интеграции с другими инструментальными средствами.

Технология создания крупных информационных систем предъявляет особые требования к методикам реализации и программным инструментальным средствам, при этом реализацию крупных проектов принято разбивать на стадии анализа (прежде чем создавать ИС необходимо понять и описать бизнес-логику предметной области), проектирования (необходимо определить модули и архитектуру будущей системы), непосредственного кодирования, тестирования и сопровождения. Известно, что исправление ошибок, допущенных на предыдущей стадии, обходится примерно в десять раз дороже, чем на текущей, откуда следует, что наиболее критичными являются первые стадии проекта. Поэтому крайне важно иметь эффективные средства автоматизации ранних этапов реализации проекта.

Большинство существующих CASE-средств основано на методологиях структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

С учетом изложенных учебно-методическое пособие посвящено характеристике современных CASE-средств моделирования предметной области автоматизации. Показаны роль и место средств ведущих компаний, а именно: описание линейки CASE средств AllFusion фирмы Computer Assotiations; Power Designer; Rational Rose.

Помимо этого, пособие содержит примеры моделей предметных областей автоматизации в Enterprise Architect.

В приложении имеется практикум «Работа с программным продуктом Enterprise Architect 9».

ГЛАВА 1. СРЕДСТВА МОДЕЛИРОВАНИЯ ПРЕДМЕТНОЙ ОБЛАСТИ АВТОМАТИЗАЦИИ

1.1. Средства описания архитектуры предприятия

Система разработки архитектуры предприятия

Описание архитектуры предприятия связано с большим количеством информации, которая поступает из разных источников (рис. 1.1) и имеет различные форматы: текст, графические модели и др. Кроме того, с этой информацией должно работать достаточно большое количество людей. Модели определенным образом связаны между собой, и эти связи необходимо также отслеживать для того, чтобы имелась возможность достаточно эффективного анализа архитектуры предприятия.

Для удовлетворения этих специфических потребностей в обеспечении архитектурного процесса появился целый класс программных продуктов. Перечислим названия некоторых компаний-разработчиков таких систем: Casewise, Computas, Framework Software, Rational Software.

Рис. 1.1. Источники информации для систем разработки архитектуры

Различные планы, модели и документы, связанные с архитектурой, необходимо поместить в репозитории (специальную базу данных, созданную для хранения многих типов документов и диаграмм) и установить между ними необходимые связи. При этом различные группы пользователей рассматривают разные категории информации и представления одной и той же архитектуры предприятия. Бизнес-менеджеров, например, больше будут интересовать диаграммы с информацией о бизнес-процессах, менеджеров ИТ – прикладные системы, связанные с этими бизнес-процессами, а на более глубоком уровне анализа – диаграммы, описывающие внутреннюю архитектуру этих систем. Если

архитектура документирована с достаточной степенью детализации, появляется возможность отслеживания влияния предполагаемых изменений. Репозиторий может содержать, например, информацию о сотрудниках и затратах, что позволяет анализировать влияние изменений в бизнес-процессах на стоимость их выполнения. Эти средства должны обеспечивать хранение артефактов в репозитории в соответствии с принятыми методиками описания архитектуры. Чем большее количество методик выбранное средство поддерживает, тем больше возможностей выбора у пользователей.

При этом многие модели и иная связанная с архитектурой предприятия информация в любом случае создается и будет создаваться с использованием других программных продуктов и инструментальных средств, поэтому средства разработки архитектуры предприятия должны иметь возможность использования внешних источников информации (рис. 1.2).

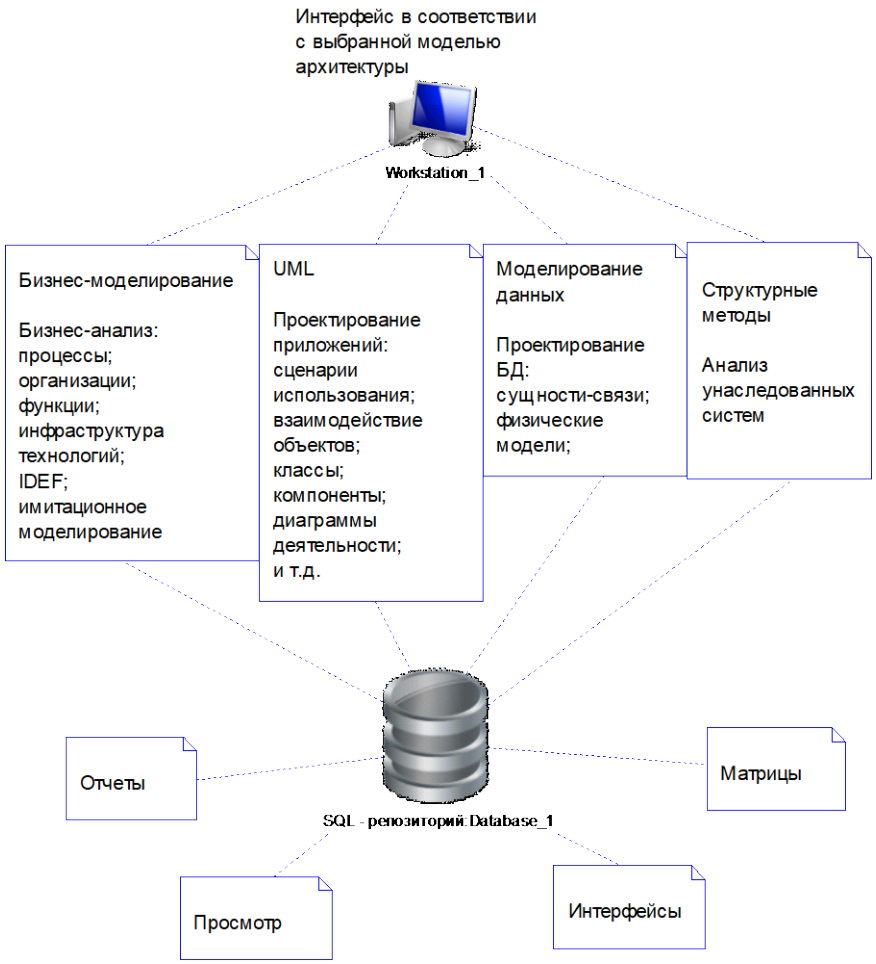


Рис. 1.2. Схема работы систем поддержки процесса разработки архитектуры

Перечислим более детально набор возможных функций систем разработки архитектуры предприятия:

поддержание списка используемых на предприятии технологий, включая версии продуктов, категории (например, СУБД, платформы, системы хранения, средства разработки и т. д.);

информационные модели бизнес-процессов, данные, функции, объекты, организационные структуры (модели «как есть» и их будущее состояние);

список прикладных систем с описанием функций, «владельцев», ответственных за эксплуатацию, поставщиков и т. д.;

кросс-ссылки. Должны быть обозначены связи прикладных систем с поддерживаемыми моделями данных и бизнес-процессов, обеспечивающими инфраструктурными технологиями;

методики описания архитектуры. Они популярны, поскольку обеспечивают способ организации огромного количества артефактов, составляющих основу описания архитектуры. Как правило, графически они отображаются в виде матриц со строками и столбцами, соответствующими различным представлениям (доменам) и уровням абстракции описания архитектуры. Должны быть обеспечены способы навигации между моделями, расположенными в различных «клетках»;

управление версиями и конфигурациями. Процесс разработки архитектуры является итерационным, включающим описания текущих и будущих состояний и моделей. Многие версии этих моделей будут часто использоваться, и ими надо управлять;

средства обеспечения полного цикла проектирования. Средства описания архитектуры предприятия должны иметь возможность обмена информацией с другими средствами проектирования и репозиториями. В идеале это должен быть двунаправленный процесс. Например, модели бизнес-процессов могут быть уже подготовлены с помощью пакета ARIS, и выбранное средство описания архитектуры предприятия должно иметь возможность обмена с этой системой;

персонифицированный доступ. Области интересов и права по работе с моделями архитектуры у различных пользователей неодинаковы;

печать и публикация. Информирование всех заинтересованных сторон – основной аспект деятельности по разработке архитектуры, поэтому важны как возможность печати достаточно сложных и больших диаграмм, так и средства доступа к этой информации с использованием, например, браузера;

географические и организационные кросс-ссылки. Разные организационные структуры отличаются используемым набором систем;

возможность настройки. Некоторые инструменты обеспечивают возможности адаптации заложенных в них стандартных архитектурных методик и моделей.

Связи между интерфейсом, который дает представление описаний архитектуры в соответствии с выбранной методикой, диаграммами, моделями и документами различных поддерживаемых форматов, репозиторием и метамоделью, а также возможностями по выводу информации на печать, просмотру и т. д., реализуемыми типичной системой поддержки разработки архитектуры, схнматично показывает рис.1.2.

CASE-средства разработки автоматизированных информационных систем

Особенности проектов информационных систем. История возникновения CASE-средств

Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности информационных систем (ИС), создаваемых в различных областях. Современные крупные проекты ИС характеризуются, как правило, следующими особенностями:

сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;

наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);

отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем;

необходимость интеграции существующих и вновь разрабатываемых приложений;

функционирование в неоднородной среде на нескольких аппаратных платформах;

разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;

существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

В 70-х и 80-х годах при разработке ИС достаточно широко применялась структурная методология, предоставляющая в распоряжение разработчиков строгие формализованные методы описания ИС и принимаемых технических решений. Она была основана на наглядной

графической технике: для описания различного рода моделей ИС используются схемы и диаграммы. Наглядность и строгость средств структурного анализа позволяли разработчикам и будущим пользователям системы с самого начала неформально участвовать в ее создании, обсуждать и закреплять понимание основных технических решений. Однако широкое применение этой методологии и следование ее рекомендациям при разработке конкретных ИС встречалось достаточно редко, поскольку при неавтоматизированной (ручной) разработке это практически невозможно. Действительно, вручную очень трудно разработать и графически представить строгие формальные спецификации системы, проверить их на полноту и непротиворечивость, и тем более изменить. Если все же удается создать строгую систему проектных документов, то ее переработка при появлении серьезных изменений практически неосуществима. Ручной разработкой обычно порождались следующие проблемы:

неадекватная спецификация требований;

неспособность обнаруживать ошибки в проектных решениях;

низкое качество документации, снижающее эксплуатационные характеристики;

затяжной цикл и неудовлетворительные результаты тестирования.

С другой стороны, разработчики ИС исторически всегда стояли последними в ряде тех, кто использовал компьютерные технологии для повышения качества, надежности и производительности.

Перечисленные факторы способствовали появлению программно-технологических средств специального класса – CASE-средств, реализующих CASE-технологию создания и сопровождения ИС. Термин CASE (*Computer Aided Software Engineering*) используется в настоящее время в весьма широком смысле. Первоначальное его значение, ограниченное вопросами автоматизации разработки только лишь программного обеспечения (ПО), в настоящее время приобрело новый смысл, охватывающий процесс разработки сложных ИС в целом. Теперь под термином «CASE-средства» понимаются программные средства, поддерживающие процессы создания и сопровождения ИС, включая анализ и формулировку требований, проектирование прикладного ПО (приложений) и баз данных, генерацию кода, тестирование, документирование, обеспечение качества, конфигурационное управление и управление проектом, а также другие процессы. CASE-средства вместе с системным ПО и техническими средствами образуют полную среду разработки ИС.

Появлению CASE-технологии и CASE-средств предшествовали исследования в области методологии программирования. Программирование обрело черты системного подхода с разработкой и внедрением языков высокого уровня, методов структурного и модульного

программирования, языков проектирования и средств их поддержки, формальных и неформальных языков описаний системных требований и спецификаций и т. д. Кроме того, появлению CASE-технологии способствовали:

подготовка аналитиков и программистов, восприимчивых к концепциям модульного и структурного программирования;

широкое внедрение и постоянный рост производительности компьютеров, позволившие использовать эффективные графические средства и автоматизировать большинство этапов проектирования;

внедрение сетевой технологии, которая способствовала объединению усилий отдельных исполнителей в процесс проектирования путем использования разделяемой базы данных, содержащей необходимую информацию о проекте.

CASE-технология представляет собой методологию проектирования ИС, а также набор инструментальных средств, позволяющих в наглядной форме моделировать предметную область, анализировать эту модель на всех этапах проектирования и сопровождения ИС и разрабатывать приложения в соответствии с информационными потребностями пользователей. Большинство существующих CASE-средств основано на методологиях структурного или объектно-ориентированного анализа и проектирования, использующих спецификации в виде диаграмм или текстов для описания внешних требований, связей между моделями системы, динамики поведения системы и архитектуры программных средств.

1.2. CASE-средства. Общая характеристика и классификация

Современные CASE-средства охватывают обширную область поддержки многочисленных технологий проектирования ИС: от простых средств анализа и документирования до полномасштабных средств автоматизации, покрывающих весь жизненный цикл ПО.

Обычно к CASE-средствам относят любое программное средство, автоматизирующее ту или иную совокупность процессов жизненного цикла ПО [32, 42] и обладающее следующими основными характерными особенностями:

мощные графические средства для описания и документирования ИС, обеспечивающие удобный интерфейс с разработчиком и развивающие его творческие возможности;

интеграция отдельных компонент CASE-средств, обеспечивающая управляемость процессом разработки ИС;

использование специальным образом организованного хранилища проектных метаданных (репозитория).

Интегрированное CASE-средство (или комплекс средств, поддерживающих полный ЖЦ ПО) содержит:

репозиторий, являющийся основой CASE-средства. Он должен обеспечивать хранение версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль метаданных на полноту и непротиворечивость;

графические средства анализа и проектирования, обеспечивающие создание и редактирование иерархически связанных диаграмм (DFD, ERD и др.), образующих модели ИС;

средства разработки приложений, включая языки 4GL и генераторы кодов;

конфигурационного управления;

документирования;

тестирования;

управления проектом;

реинжиниринга.

Программные средства поддержки жизненного цикла автоматизированных систем. Классификация CASE-средств

Можно привести много примеров различных классификаций CASE-средств, встречающихся в литературе. Рассмотрим классификацию современных CASE-систем:

1. По ориентации на процессы ЖЦ ПО:

средства анализа и проектирования (BPWin, Silverrun, Oracle Designer, Rational Rose, Paradigm Plus, Power Designer, System Architect);

средства проектирования баз данных (средства проектирования баз данных имеются в составе таких CASE-средств, как Silverrun, Oracle Designer, Paradigm Plus, Power Designer. Наиболее известным средством, ориентированным только на проектирование БД, является ERWin);

средства управления требованиями (RequisitePro, DOORS – Dynamic Object-Oriented Requirements System – динамическая объектно-ориентированная система управления требованиями);

средства управления конфигурацией ПО (PVCS, ClearCase и др.);

средства документирования. (SoDA – Software Document Automation – автоматизированное документирование ПО);

средства тестирования. (Rational Suite TestStudio);

средства управления проектом (Open Plan Professional, Microsoft Project и др.);

средства реверсного инжиниринга, предназначенные для переноса существующей системы ПО в новую среду. Средства анализа схем БД и формирования ERD входят в состав таких CASE-средств, как Silverrun, Oracle Designer, Power Designer, ERwin. Анализаторы программных кодов имеются в составе Rational Rose и Paradigm Plus.

2. По поддерживаемым методологиям проектирования:

Функционально- (структурно-) ориентированные;

объектно-ориентированные;

комплексно-ориентированные (набор методологий проектирования).

3. По поддерживаемым графическим нотациям построения диаграмм:

с фиксированной нотацией;

с отдельными нотациями;

с наиболее распространенными нотациями.

4. По степени интегрированности:

tools (отдельные локальные средства);

toolkit (набор неинтегрированных средств, охватывающих большинство этапов разработки ИС);

workbench (полностью интегрированные средства, связанные общей базой проектных данных – репозиторием).

5. По типу и архитектуре вычислительной техники:

ориентированные на ПЭВМ;

ориентированные на локальную вычислительную сеть (ЛВС);

ориентированные на глобальную вычислительную сеть (ГВС);

смешанного типа;

6. По режиму коллективной разработки проекта:

не поддерживающие коллективную разработку;

ориентированные на режим реального времени разработки проекта;

ориентированные на режим объединения подпроектов.

7. По типу операционной системы:

работающие под управлением WINDOWS 3.11 и выше;

работающие под управлением UNIX;

работающие под управлением различных ОС (WINDOWS, UNDO, OS/2 и др.).

8. По типам – отражает функциональную ориентацию CASE-средств в технологическом процессе:

анализ и проектирование. Средства этой группы используются для создания спецификаций системы и ее проектирования: они поддерживают широкоизвестные методологии проектирования. К таким средствам относятся The Developer (Asyst Technologies), Design Generator (Computer Sciences). Pose (Computer Systems Advises). Analisis/Designer (Jour-don) и т.д.;

проектирование баз данных и файлов. Средства обеспечивают логическое моделирование данных, генерацию схем БД и описание форматов файлов: Erwin, PowerDesigner, Idef/Leverage (D.Appleton), Chen Toolkit (CTien & Associates). Case+Designer (Orale) и т. д.;

программирование. Средства поддерживают шаги программирования и тестирования, а также автоматическую

кодогенерацию из спецификаций, получая полностью документированную выполняемую программу: Miero Focus, Decase (DEC), Netron/Cap (Netron) и др;

сопровождение и реинжинерия. К таким средствам относятся документаторы, анализаторы программ, (средства реструктурирования и обратной инжинерии: Adpac Case Tools (Adpac), Superstructure (Computer Data Systems) и т. д.;

окружение. Средства, поддерживающие платформы для интеграции, создания и придания товарного вида CASE-средствам: Multi/Cum (ACiS Management Systems), Sylvia Foondey (Codinare);

управление проектом. Средства, поддерживающие планирование, контроль, руководство, взаимодействие, т. е. функции. необходимые в процессе разработки и сопровождения проектов, например, Project Workbench (Applied Business Technology).

9. По категориям – определяет уровень интегрированности по выполняемым функциям и включает:

вспомогательные программы (Tools), решающие небольшую автономную задачу, принадлежащую проблеме более широкого масштаба;

пакеты разработки (Toolkit), представляющие собой совокупность интегрированных средств, обеспечивающих помощь для одного из классов программных задач;

инструментальные средства (Workbench) по сравнению с Toolkit обладают более высокой степенью интеграции выполняемых функций, большей автономностью использования. Они тесно связаны с системными и техническими средствами аппаратно-вычислительной среды, на которой функционируют. Workbench – это автоматизированная рабочая стадия, используемая как инструментарий для автоматизации всех или отдельных совокупностей работ по созданию ПО АС.

10. По уровням – связана с областью действия CASE в пределах жизненного цикла ПО.

Верхние (Upper) CASE часто называют *компьютерным планированием*. Их использование позволяет построить модель предметной области, которая отражает всю существующую специфику и направлена на понимание общего и частного механизмов функционирования, имеющихся возможностей, ресурсов и целей проекта в соответствии с назначением фирмы. Эти средства позволяют проводить анализ различных сценариев, накапливая информацию для принятия оптимальных решений.

Средние (Middle) CASE считаются средствами поддержки этапов анализа требований и проектирования спецификаций и структуры ИС. Основная выгода от использования среднего CASE состоит в значительном облегчении проектирования систем. Проектирование превращается в итеративный процесс, включающий действия:

пользователь обсуждает с аналитиком требования к информации; аналитик документирует эти требования, используя диаграммы и словари входных данных; пользователь проверяет эти диаграммы и словари, при необходимости модифицируя их; аналитик отвечает на эти модификации, изменяя соответствующие спецификации. Кроме того, средние CASE обеспечивают возможности быстрого документирования требований и прототипирования.

Нижние (Lower) CASE поддерживают системы разработки ПО ИС (при этом может использоваться до 30 % спецификаций, созданных средствами среднего CASE). Главными преимуществами нижних CASE является: значительное уменьшение времени на разработку, облегчение модификаций, поддержка возможностей прототипирования (совместно со средними CASE).

На сегодняшний день российский рынок программного обеспечения располагает практически всеми перечисленными выше средствами. Наиболее популярные CASE-средства проектирования ИС приведены в табл. 1.1. Отметим, что многие из этих продуктов предназначены для решения разнообразных задач, например, моделирования потоков данных или бизнес-процессов, функционального моделирования, проектирования данных, прототипирования приложений, их документирования, управления проектами и т. д.

Таблица 1.1. Наиболее популярные CASE-средства

CASE-средство	Производитель	Адрес сайта производителя
Designer 2000	Oracle	http://www.oracle.com/
ERwin/BPwin	Computer Associates	http://www.cai.com/
PowerDesigner	Sybase	http://www.sybase.com/
ER/Studio	Embarcadero	http://www.embarcadero.com/
System Architect	Popkin Software	http://www.popkin.com/
Visible Analyst	Visible Systems	http://www.visible.com/
Visio Enterprise	Microsoft	http://www.microsoft.com/

1.3. Описание линейки CASE средств AllFusion фирмы Computer Assotiations

Линейка *AllFusion* компании Computer Associates – это семейство интегрированных решений для разработки, развертывания и управления информационными системами на предприятии. Средства моделирования и инструменты управления изменениями и конфигурациями при проектировании ПО позволяют организациям моделировать, разрабатывать и внедрять информационные системы масштаба предприятия.

AllFusion Modeling Suite – линейка интегрированных средств моделирования (CASE)

CASE-средства Computer Associates позволяют моделировать бизнес-процессы, базы данных, компоненты программного обеспечения, деятельность и структуру организаций. Закономерный результат применения CASE-средств – оптимизация систем, снижение расходов, повышение эффективности, снижение вероятности ошибок и т. д.

Краткая характеристика программных продуктов, входящих в Suite

AllFusion Process Modeler – инструмент визуального моделирования бизнес-процессов. Дает возможность наглядно представить любую деятельность или структуру в виде модели, что позволит оптимизировать работу организации, проверить ее на соответствие стандартам ISO9000, спроектировать структуру организации, снизить издержки, исключить ненужные операции, повысить гибкость и эффективность. BPwin поддерживает сразу три нотации моделирования: IDEF0 (федеральный стандарт США), IDEF3 и DFD.

AllFusion ERwin Data Modeler – позволяет проектировать, документировать и сопровождать базы данных, хранилища данных и витрины данных (data marts). Создав наглядную модель базы данных, можно оптимизировать структуру БД и добиться ее полного соответствия требованиям и задачам организации. Визуальное моделирование повышает качество создаваемой базы данных, продуктивность и скорость ее разработки.

AllFusion Data Model Validator – инструмент для проверки структуры баз данных и создаваемых в ERwin моделей, позволяющий выявлять недочеты и ошибки проектирования. ERwin Examiner дополняет функциональность ERwin, автоматизируя трудоемкую задачу поиска и исправления ошибок, одновременно повышая квалификацию проектировщиков баз данных благодаря встроенной системе обучения.

AllFusion Model Manager – среда для совместной работы группы проектировщиков на AllFusion ERwin Data Modeler *(ERwin)* и/или AllFusion Process Modeler *(BPwin)* над одним проектом. Обеспечивает совместный доступ к моделям, возможность их редактирования, повышая эффективность и скорость работы проектировщиков. Является интегрирующим звеном для ERwin (моделирование баз данных) и BPwin (моделирование бизнес-процессов). Защищает данные, хранимые на собственном сервере модели, позволяет задавать для сотрудников различный уровень доступа к ним и координировать весь ход работы над проектом.

AllFusion Component Modeler – CASE-средство для проектирования, визуализации и поддержки качественных информационных систем. Благодаря обеспечению расширенной поддержки совместного проектирования и многократного использования компонентов модели продукт можно использовать как при создании новых приложений, так и

при изменении или объединении существующих. Благодаря интеграции с *AllFusion Process Modeler (BPwin)* есть возможность использования функциональной модели вместе с объектной. Продукт поддерживает около десятка стандартных нотаций, таких как UML и Booch, интегрируется с технологиями COM/DCOM, CORBAPlus, BES VisiBroker и др., продуктами CA, Microsoft, Rational Software и др.

AllFusion Model Navigator – инструмент для просмотра моделей, созданных в AllFusion Process Modeler и AllFusion ERwin DataModeler, в режиме «только для чтения». Поддерживает функции просмотра моделей, их печати, а также редактирования их оформления и позволяет сотрудникам, не занимающимся напрямую разработкой моделей, пользоваться информацией, содержащейся в них. Это помогает предотвратить несанкционированные изменения моделей, но при этом использовать их для создания презентаций, разработки приложений и т. д.

Выводы о возможностях продукта BPwin

Автоматизация процесса проектирования. BPwin автоматизирует многие задачи, обычно связанные с построением моделей процессов, обеспечивая семантическую точность, необходимую для гарантии правильных и согласованных результатов. Подсветка объектов упрощает построение модели, исключая часто встречающиеся ошибки моделирования.

Свойства, определяемые пользователем. Можно настроить BPwin для сбора информации, существенной для конкретного бизнеса. Эта информация становится сразу же доступной через генератор отчетов BPwin и может быть экспортирована в другие программы, например, Microsoft Word и Excel.

Диаграммы Swim Lane. BPwin поддерживает диаграммы Swim Lane, предоставляя эффективный механизм для визуализации и оптимизации сложных бизнес-процессов. Диаграммы Swim Lane координируют сложные процессы и функциональные ограничения и позволяют видеть процессы, роли и обязанности во всем их многообразии.

Развитые диаграммы. К таким диаграммам отнисятся:

контекстные диаграммы для описания границ системы, области действия, назначения объектов, отличающиеся иерархической структурой, облегчающей последовательное уточнение элементов модели;

декомпозиционные диаграммы для описания особенностей взаимодействия различных процессов.

BPwin также поддерживает автоматическую настройку размеров диаграмм и возможность изменения масштабов изображения моделей.

Организационные диаграммы – оказывают огромное влияние на выделение и выполнение бизнес-процессов. BPwin поддерживает явное определение ролей, а это категоризирует задачи или работы,

составляющие бизнес-процессы. Основываясь на ролях, определенных пользователем, BPwin формирует организационные диаграммы.

Технологии моделирования. BPwin обеспечивает совместное и повторное использование технологий моделирования бизнес-процессов (IDEF0), потоков работ (IDEF3) и потоков данных (DFD).

Функционально-стоимостной анализ (*ABC*). BPwin полностью поддерживает методы расчета себестоимости по объему хозяйственной деятельности (ABC) и оптимизирована для анализа процессов. Развитые средства подготовки отчетов и двунаправленный интерфейс со специализированным инструментарием ABC облегчают реализацию корпоративной стратегии на основе управления хозяйственной деятельностью.

Собственный генератор отчетов. Report Template Builder (RTB) – это новый генератор отчетов, общий для ERwin и BPwin, который создает разнообразные отчеты и Web-страницы. Можно определять шаблоны отчетов, применяя их затем к любым своим моделям. Подход «определить однажды – применять повторно и повсюду» позволяет организации быстро создавать и продвигать стандарты отчетности. RTB поддерживает множество форматов, включая RTF, HTML, XLS (Excel) и обычный текст.

Интерфейс к средствам имитационного моделирования. Для моделирования сложных условий деятельности BPwin предлагает интерфейс к имитационному ПО (например, Arena). Это позволяет использовать готовые модели для изучения взаимодействия бизнес-процессов, изменяющегося во времени (динамического). Распределение ресурсов и потоки могут быть оптимизированы для достижения эффективной загрузки. Имитационное моделирование позволяет в динамике проанализировать воздействие изменений. Прежде чем эти изменения будут произведены, можно проверить различные сценарии и обеспечить тем самым принятие оптимального решения.

Характеристика AllFusion ERwin Data Modeler (ERwin)

Разработка в среде ERwin

Обычно разработка модели базы данных состоит из двух этапов: составления логической модели и создания на ее основе физической модели. ERwin полностью поддерживает такой процесс, он имеет два представления модели: логическое (logical) и физическое (physical). Таким образом, разработчик может строить логическую модель базы данных, не задумываясь над деталями физической реализации, т. е. уделяя основное внимание требованиям к информации и бизнес-процессам, которые будут поддерживаться создаваемой базой данных. ERwin имеет очень удобный пользовательский интерфейс, позволяющий представить базу данных в различных аспектах. Предметные области помогают вычленить из сложной и трудной для восприятия модели отдельные фрагменты, которые

относятся лишь к определенной области, из числа тех, что охватывает информационная модель.

Возможность использования модели ERwin одновременно для логического и физического представления данных позволяет по окончании работы получить полностью документированную модель. Документирование структуры данных является очень важной частью моделирования, так как дает возможность другим разработчикам или лицам, которые будут сопровождать систему, быстрее ориентироваться во внутренней ее структуре и понимать назначение компонентов.

Как уже говорилось, ERwin является не только инструментом для дизайна баз данных, но и поддерживает автоматическую генерацию спроектированной и определенной на физическом уровне структуры данных. ERwin поддерживает широчайший спектр серверных и настольных СУБД. В этот список входят такие продукты, как Microsoft SQL Server, Oracle, Sybase, DB2, INFORMIX, Red Brick, Teradata, PROGRESS, Microsoft Access, FoxPro, Clipper и многие другие. Для каждой из перечисленных СУБД в ERwin предусмотрено присоединение по специфическому протоколу и поддержка всех средств управления данными. ERwin имеет средство, выполняющее обратную генерациию, что называется «обратная разработка» (reverse engineering): ERwin может присоединиться к СУБД, получить всю информацию о структуре базы данных и отобразить ее в графическом интерфейсе, сохранив все сущности, связи, атрибуты и прочие свойства. Таким образом, можно переносить существующую структуру данных с одной платформы на другую, а также исследовать структуру существующих баз данных.

ERwin имеет средство Complete-Compare, являющееся единственным на данный момент способом интерактивной разработки. С его помощью все изменения модели можно вносить в базу данных автоматически без необходимости контроля за соответствием модели и базы данных «вручную», при этом существующие данные не будут затронуты. Таким образом, ERwin демонстрирует разногласия между моделью и базой данных, которые можно переносить или оставлять без изменений.

ERwin поддерживает многомерное моделирование, используемое при построении хранилищ данных. Производительность OLAP-приложений определяется в основном качеством дизайна хранилища данных, поэтому критически важно при разработке хранилища иметь инструмент, который бы способствовал работе распространенных технологий. ERwin дает возможность применять две технологии моделирования хранилищ данных: звезда (star) и снежинка (snowflake).

ERwin тесно интегрирован с другими продуктами CA/Logic Works. Словарь данных, созданный при анализе бизнес-процессов с помощью инструмента BPwin, может быть использован как основа для построения модели базы данных. Однако взаимосвязь между этими двумя

инструментами двусторонняя, модели BPwin и ERwin можно постоянно поддерживать в согласованном состоянии. Интеграция этих двух продуктов очень важна с точки зрения их совместного использования при разработке программного обеспечения, так как отпадает необходимость в повторном выполнении действий и процесс создания словаря данных становится практически автоматическим.

Поддерживаемые нотации

ERwin поддерживает стандартную нотацию *IDEF1x* для ER-диаграмм моделей данных, нотацию IE и специальную нотацию, предназначенную для проектирования хранилищ данных – *Dimensional.*

1.4. Разработка баз данных с ERwin DM

Начало создания модели в AllFusion ERwin DM

Для создания новой модели выбирают инструмент  на стандартной панели инструментов или команду New в меню File. В результате открывается диалоговое окно Create Model – Select Template (рис. 1.3).

В разделе New Model Type отмечают один из трех возможных типов новой модели: Logical (логическая), Physical (физическая) или Logical/Physical смешанная модель. В разделе Create Using Template отображается название шаблона, на основе которого будет создана новая модель. Шаблоном, используемым по умолчанию, является Blank Model (пустая модель). Подключают другой шаблон с помощью кнопок Browse File System или Browse AllFusion MM. В первом случае шаблон модели должен находиться в файле с расширением *.erwin_tmpl, во втором – в репозитории AllFusion Model Manager.

В ERwin DM шаблон создается из модели данных и используется как основа для быстрого создания новых моделей данных. Чтобы сохранить существующую модель данных как шаблон в файл с расширением *.erwin_tmpl, следует выбрать меню File/Save As.

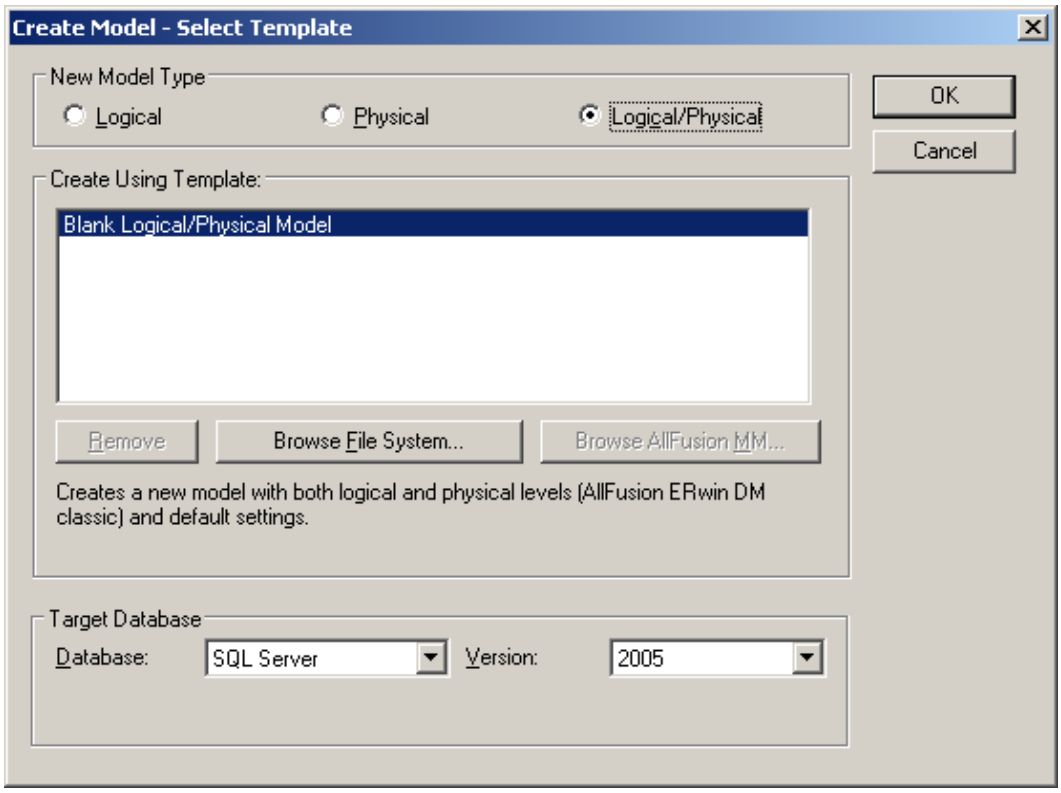


Рис. 1.3. Диалог Create Model – Select Template

В случае, когда новая модель определена как физическая или смешанная, требуется найти целевую СУБД в разделе Target Database: в выпадающем списке Database выбирают название СУБД, а в списке Version – ее версию. Если новая модель определена как логическая, целевую СУБД выбирать не нужно. После определения параметров новой модели диалоговое окно Create Model – Select Template можно закрыть, нажав на кнопку ОК. В результате будет создана модель с именем по умолчанию *Model_i*, где *i* – номер модели, назначаемый ERwin DM автоматически. Имя модели отображается в заголовке окна и в навигаторе модели; текущим уровнем модели по умолчанию – логический (рис. 1.4).

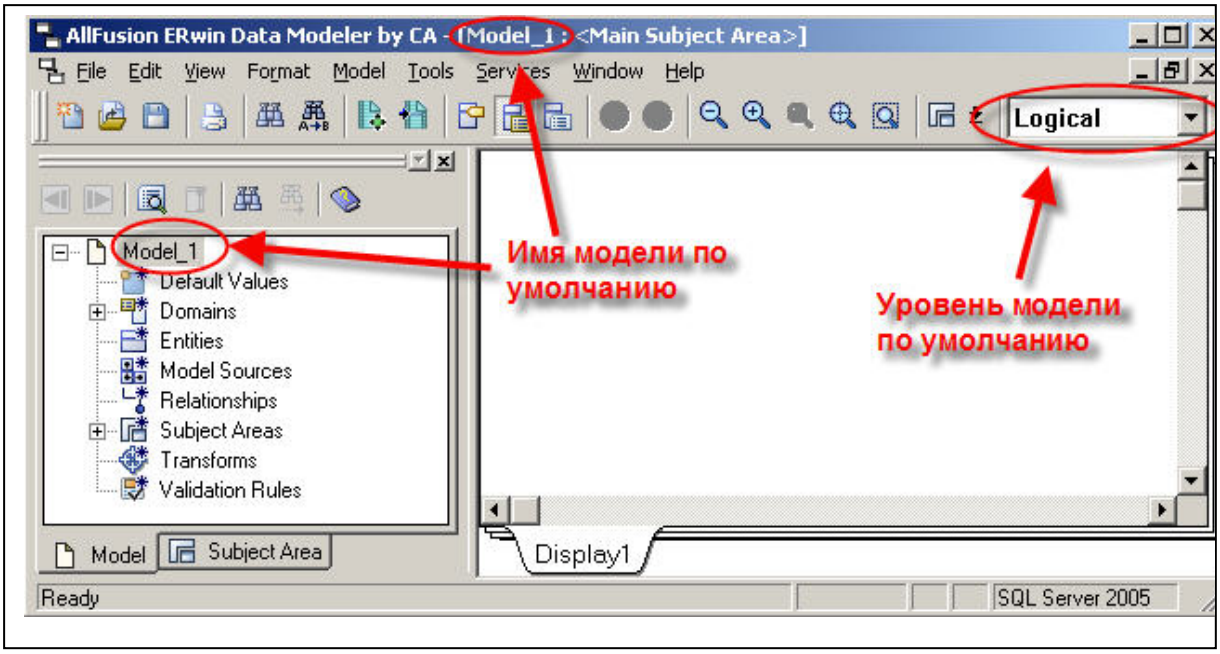


Рис. 1.4. Имя и уровень новой модели по умолчанию

В самом начале работы с моделью рекомендуется настроить рабочую область (см. раздел «Инструментальная среда AllFusion ERwin DM»): скрыть или отобразить необходимые для работы панели инструментов, навигатор модели. Кроме того, перед началом моделирования рекомендуется настроить свойства модели в диалоге Model Properties (меню Model/Model Properties).

Уровни модели данных

Различают 3 подуровня логического уровня модели данных, отличающиеся по глубине представления информации о данных, и 2 подуровня физического уровня (рис. 1.5).

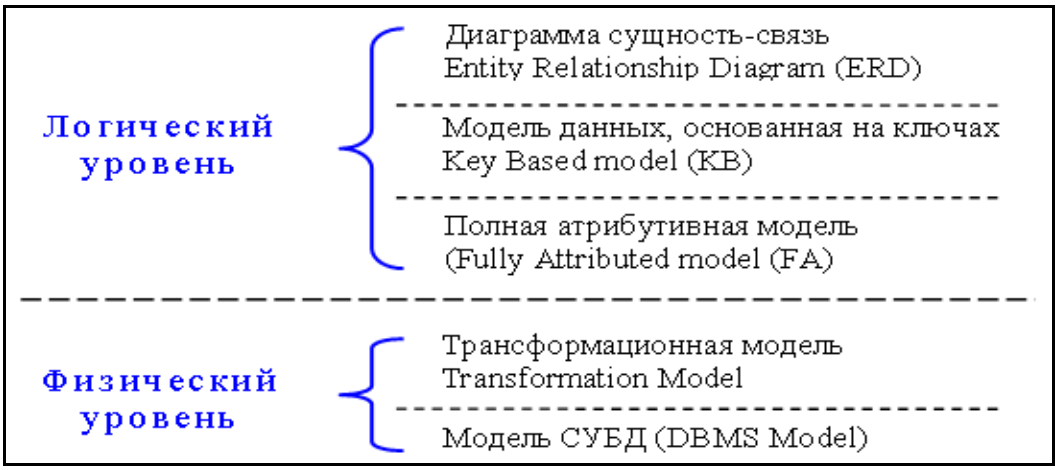


Рис. 1.5. Уровни модели данных

Диаграмма сущность-связь включает сущности и взаимосвязи, отражающие основные бизнес-правила предметной области. Такая диаграмма не слишком детализирована, в нее включаются основные сущности и связи между ними, которые удовлетворяют основным требованиям, предъявляемым к информационной системе. Может включать связи «многие ко многим» при отсутствии описания ключей. Как правило, ERD используется для презентаций и обсуждения структуры данных с экспертами предметной области.

Модель данных, основанная на ключах – более подробное представление данных. Включает описание всех сущностей и первичных ключей и предназначена для представления структуры данных и ключей, которые соответствуют предметной области.

Полная атрибутивная модель – наиболее детальное представление структуры данных: отображает данные в третьей нормальной форме и включает все сущности, атрибуты и связи.

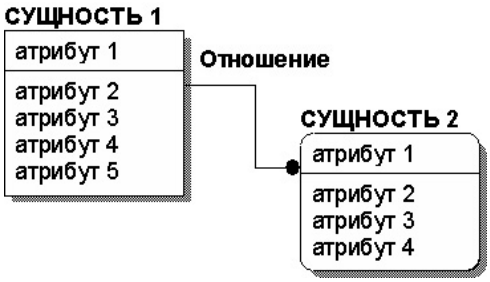
Трансформационная модель содержит информацию для реализации отдельного проекта, который может быть частью общей ИС и описывать подмножество предметной области. ERwin DM поддерживает ведение отдельных проектов, позволяя проектировщику выделять подмножество модели в виде предметных областей (Subject Area). Трансформационная модель позволяет проектировщикам и администраторам баз данных лучше представлять, какие объекты базы данных хранятся в словаре данных, и проверить, насколько физический уровень модели данных удовлетворяет требованиям к ИС.

Модель СУБД автоматически генерируется из трансформационной модели и является точным отображением системного каталога СУБД. ERwin DM непосредственно поддерживает ее путем генерации системного каталога.

Создание логического уровня модели

Основными компонентами диаграммы логического уровня модели в ERwin DM являются сущности, атрибуты, связи (отношения) (рис. 1.6).

Сущность – множество подобных индивидуальных объектов, называемых *экземплярами*. *Атрибут* выражает определенное свойство объекта. Построение модели данных предполагает определение сущностей и атрибутов: необходимо определить, какая информация будет храниться в конкретной сущности и в конкретном атрибуте. На физическом уровне сущности соответствует таблица, экземпляру сущности – строка в таблице, а атрибуту – колонка таблицы (рис. 1.7).



На логическом уровне

- Сущности
- Атрибуты
- Отношения

Рис. 1.6. Базовые объекты модели логического уровня

Логический уровень	Физический уровень
ЗАКАЗЧИК Номер заказчика Фамилия заказчика Адрес заказчика	CUSTOMER Customer_number Customer_name Customer_address
а) Сущность	б) Таблица

Рис. 1.7. Пример сущности и соответствующей таблицы

Сущности

Сущность можно определить как объект, событие или концепцию, информация о которых должна сохраняться. Она должна иметь наименование с четким смысловым значением, которое является существительным в единственном числе и не относится к «техническим», а также быть достаточно важной для моделирования. Именование сущности в единственном числе облегчает в дальнейшем чтение модели. Фактически имя сущности дается по имени ее экземпляра. Примером может быть сущность *Заказчик* (но не *Заказчики!)* с атрибутами *Номер заказчика*, *Фамилия заказчика* и *Адрес заказчика*. На уровне физической модели ей может соответствовать таблица *Customer* с колонками *Customer_number*, *Customer_name* и *Customer_address* (рис. 1.8).

Для внесения сущности в модель необходимо (это делается на уровне логической модели) щелкнуть на кнопке  на панели инструментов ERwin Toolbox, затем – по тому месту диаграммы, где планируется расположить новую сущность. Далее, щелкнув правой кнопкой мыши по сущности и выбрав из контекстного меню пункт Entity Properties, можно вызвать диалог Entities, в котором определяются имя, описание и комментарии сущности (рис. 1.9).

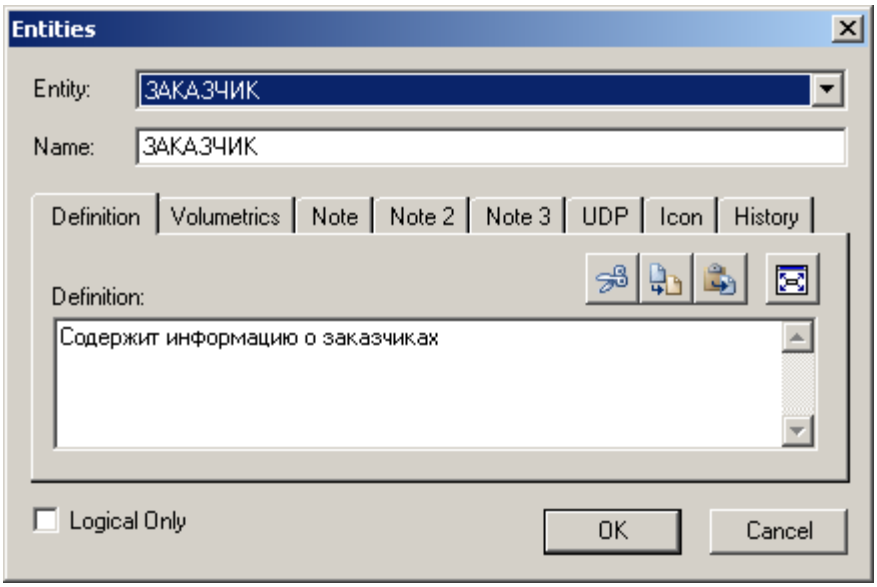


Рис. 1.8. Закладка Definition диалога Entities

Каждая сущность должна быть полностью определена с помощью текстового описания в закладке Definition. Закладки Note, Note 2, Note 3, UDP (User Defined Properties – свойства, определенные пользователем) служат для внесения дополнительных комментариев и определений к сущности. В ранних версиях ERwin закладкам Note2 и Note3 соответствовали окна Query и Sample.

Закладка *Definition* (см. рис. 1.8) используется для ввода определения сущности. Эти определения полезны как на логическом уровне (позволяют понять, что это за объект), так и на физическом уровне (их можно экспортировать как часть схемы базы данных и использовать в реальной базе данных). При соответствующих настройках генерации схемы ERwin DM автоматически сгенерирует скрипт *CREATE COMMENT on entity_name*.

Закладка *Volumetrics* (рис. 1.9) позволяет на логическом уровне вводить информацию о приблизительном размере соответствующих таблиц. Для оценки размера таблицы вводят следующую информацию:

Initial Rows – начальное количество строк в таблице,

Max Rows – максимальное число (лимит) строк в таблице,

Grow By – скорость увеличения таблицы (строк в месяц).

Закладка *Note* позволяет вносить дополнительные замечания о сущности, которые не были отражены в определении, введенном в закладке Definition. Например, сюда вводится полезное замечание, описывающее бизнес-правило или соглашение по организации диаграммы.

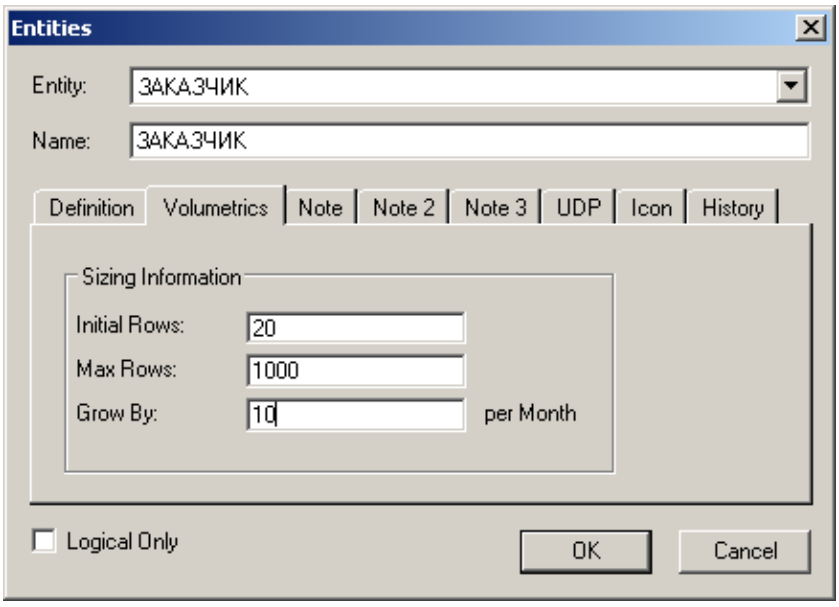


Рис. 1.9. Закладка Volumetrics диалога Entities

В закладке *Note 2* можно задокументировать возможные запросы, которые, как ожидается, будут использоваться по отношению к конкретной сущности в базе данных. При переходе к физическому проектированию записанные в закладке Note 2 примеры запросов помогут сделать базу данных более эффективной.

Закладка *Note 3* позволяет в произвольной форме вводить примеры экземпляров сущности.

Применение свойств, определяемых пользователем (UDP), аналогично использованию в AllFusion Process Modeler. Для определения UDP служит диалог User Defined Properties (меню Model/UDP Dictionary) (рис. 1.10). В нем необходимо указать вид объекта, для которого заводится UDP (диаграмма в целом, сущность, атрибут и т. д.), и тип данных. Для внесения нового свойства следует ввести имя, тип данных, значение по умолчанию и описание. Следующая строка таблицы появляется автоматически.

ERwin DM поддерживает для UDP шесть типов данных:

Date. Дата. Применяется формат MM/DD/YY. Для выбора значения даты можно использовать контекстный календарь.

Int. Целое число.

Real. Действительное число.

Text. Строка (ASCII).

List. Список. При задании списка в диалоге User Defined Property значения следует разделять запятой, значение по умолчанию выделяется символом ~ (тильда).

Command. Команда – выполняемая строка. На рис. 1.10 свойство *Документ* имеет тип *Command.*

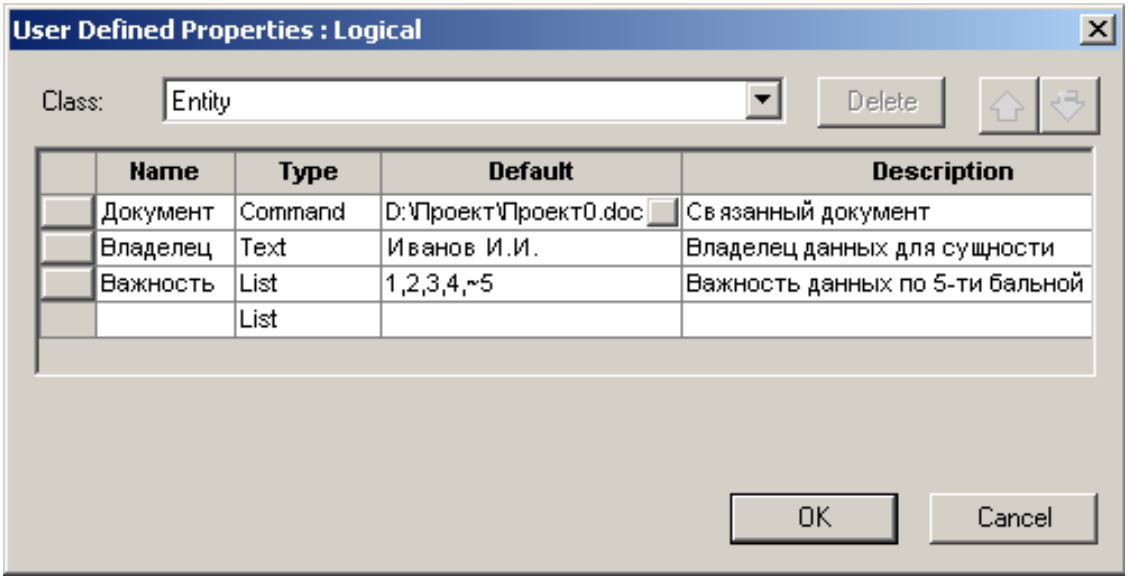


Рис. 1.10. Диалог User Defined Properties

Значение свойств, определяемых пользователем, задается в закладке *UDP* диалога Entities (рис. 1.11). Если пользовательскому свойству *Документ* присвоить значение «*D:\Проект\Проект0.doc*», то в модели из закладки UDP можно редактировать файл «*Проект0.doc*» (кнопка в строке свойства *Документ*). Если у сущности требуется изменить значение свойства типа List, например, свойство *Важность*, можно либо ввести одно из допустимых значений в строке ввода, либо просто выбрать его из списка допустимых значений. В последнем варианте следует сначала щелкнуть мышкой по строке ввода значения свойства, затем по появившемуся значку и в выпавшем списке выбрать требуемое значение свойства.

С помощью закладки *Icon* каждой сущности можно поставить в соответствие картинку, которая будет отображаться в режиме просмотра модели на уровне иконок. В этой закладке задается как большая иконка, отображаемая на уровне Icon, так и малая, которая может отображаться на всех уровнях просмотра модели. Для связывания изображения с сущностью необходимо щелкнуть по кнопке , в появившемся диалоге Icons – по кнопке Import и выбрать соответствующий файл формата BMP. После выбора иконки она отображается во вкладке Icon диалога Entities (рис. 1.12).

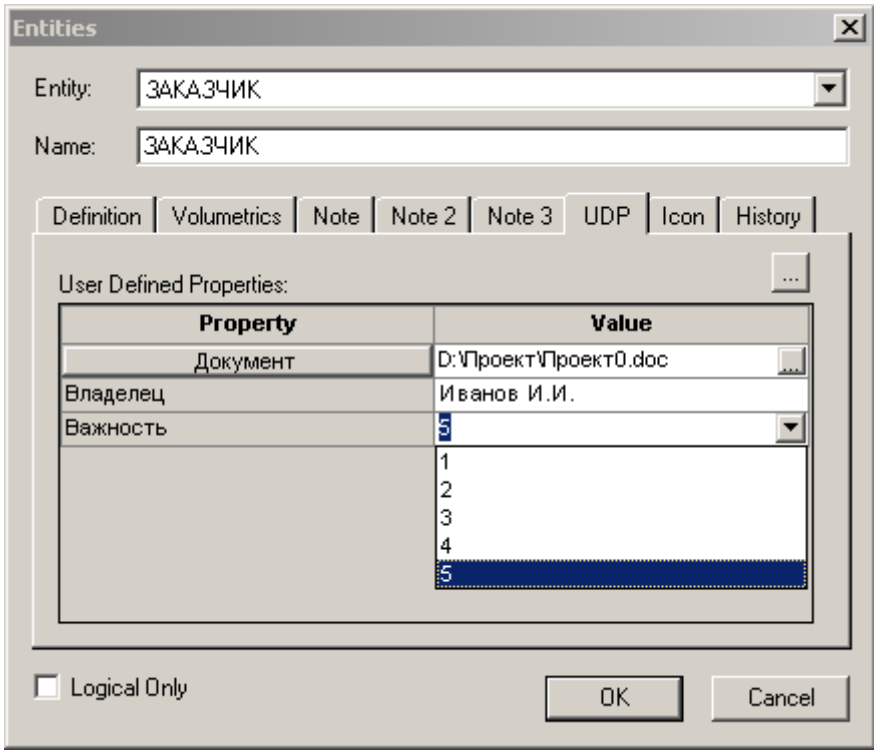


Рис. 1.11. Закладка UDP диалога Entities

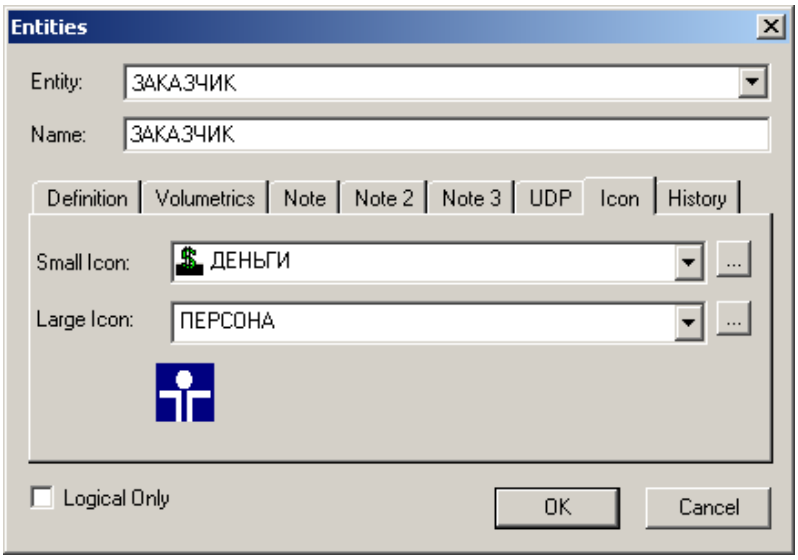


Рис. 1.12. Закладка Icon диалога Entities

ERwin DM автоматически сохраняет историю всех изменений, связанных с объектами (сущностями, атрибутами, таблицами, колонками и т. д.). В закладке *History* диалога Entities (рис. 1.13) отображается список изменений. Каждому изменению в окне Comment можно дать комментарий.

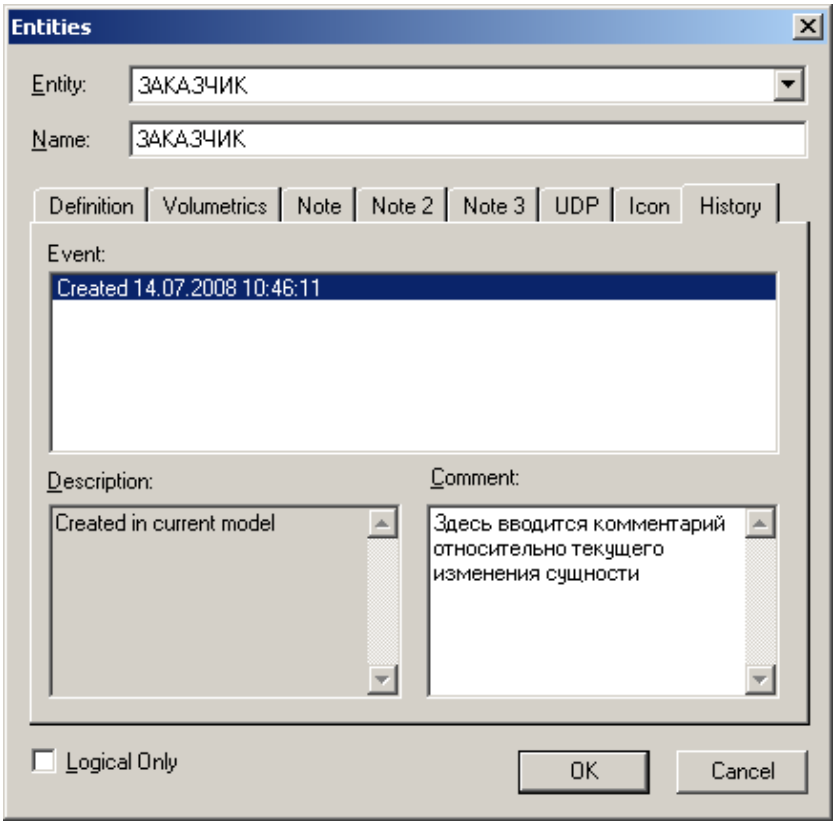


Рис. 1.13. Закладка History диалога Entities

Атрибуты

Как было указано выше, каждый атрибут хранит информацию об определенном свойстве сущности, а каждый экземпляр сущности должен быть уникальным. Атрибут или группа атрибутов, которые идентифицируют сущность, называют *первичным ключом*. Для описания атрибутов следует, щелкнув правой кнопкой по сущности, выбрать в появившемся меню пункт Attributes. Появляется диалог Attributes (рис. 1.14).

Если щелкнуть по кнопке New, то в появившемся диалоге New Attribute можно указать имя атрибута, имя соответствующей ему в физической модели колонки и домен. Домен атрибута будет использоваться при определении типа колонки на уровне физической модели.

Для атрибутов первичного ключа в закладке *General* диалога Attributes необходимо сделать пометку в окне выбора Primary Key. Для наглядности диаграммы каждый атрибут можно связать с иконкой из выпадающего списка Icon в закладке General. Если имеющихся в списке графических изображений недостаточно, следует воспользоваться кнопкой ..., расположенной справа от списка выбора. В результате откроется диалог Icons (рис. 1.15). Щелкнув по кнопке Import, можно добавить в список необходимую иконку. Чтобы отобразить на диаграмме иконки атрибутов, нужно, во-первых, перейти на уровень атрибутов (меню

Format/Display Level/Attribute), во-вторых, установить режим отображения иконок для атрибутов (меню Format/Entity Display/Attribute Icon).

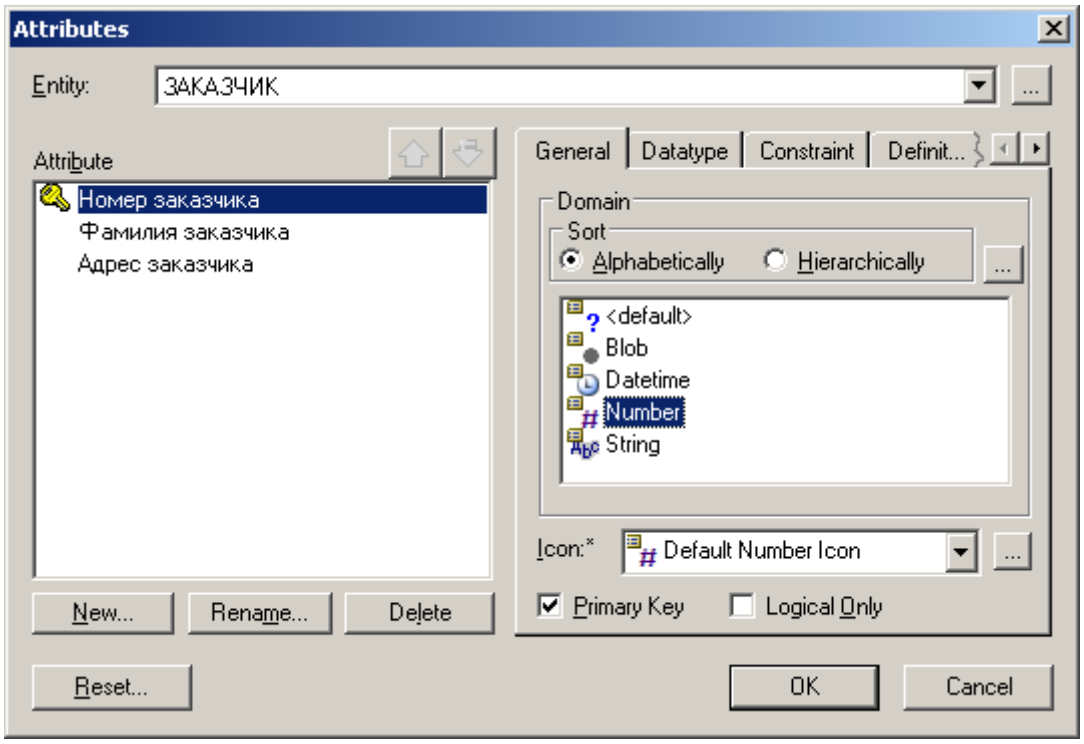


Рис. 1.14. Закладка General диалога Attributes

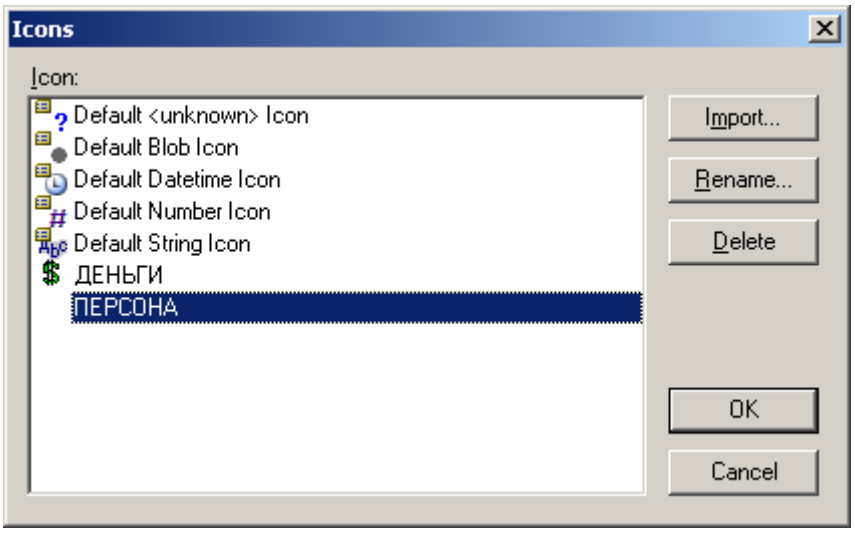


Рис. 1.15. Диалог Icons

Закладка *Definition* диалога Attributes позволяет записывать определения отдельных атрибутов. Определения атрибутов можно также автоматически сгенерировать как часть схемы (CREATE COMMENT on entity_name.attribute name).

Закладка *Note* позволяет добавлять замечания об одном или нескольких атрибутах сущности, которые не вошли в определения.

В закладке *History* отображается история создания и изменения атрибутов.

Закладка *UDP* служит для задания значений свойств, определяемых пользователем. Предварительно эти свойства должны быть внесены в диалог User Defined Property как свойства атрибутов.

Закладка *Key Group* (рис. 1.16) позволяет включить атрибут в состав первичного, альтернативного или инверсного ключа.

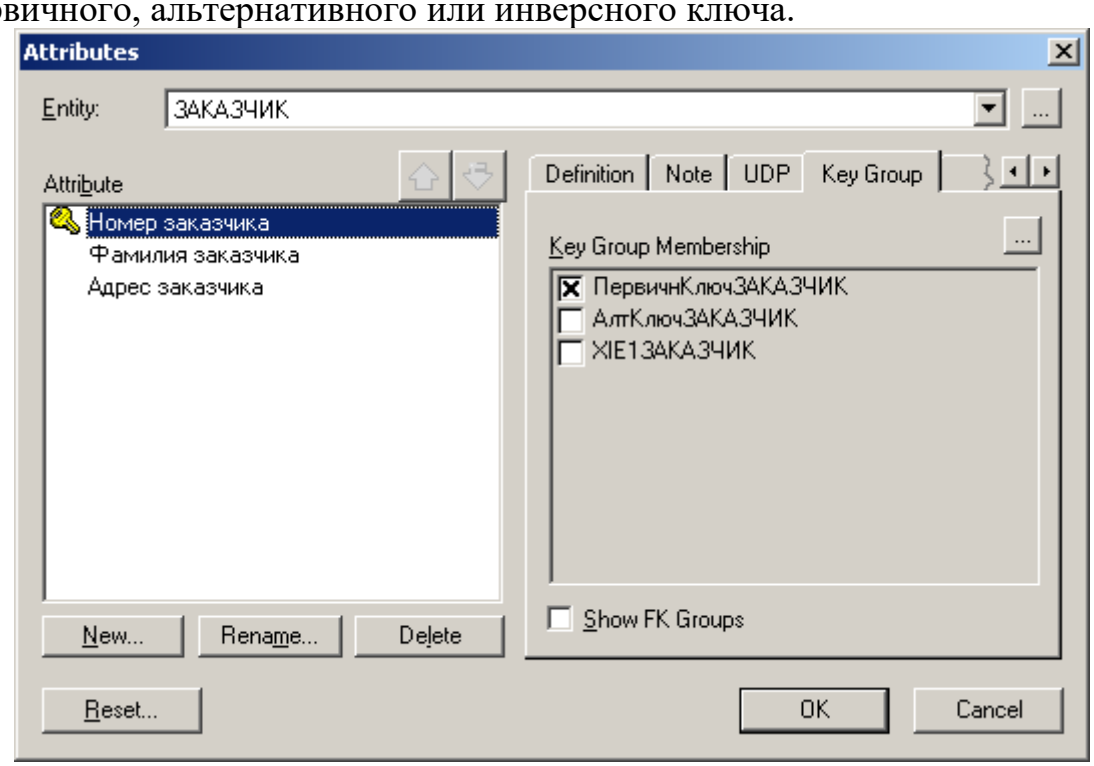


Рис. 1.16. Закладка Key Group диалога Attributes

На диаграмме IDEF1X сущность и атрибуты отображаются следующим образом: имя сущности показывается над прямоугольником, изображающим сущность, список атрибутов сущности – внутри прямоугольника. Список разделен горизонтальной чертой, выше которой расположены атрибуты первичного ключа, ниже – неключевые атрибуты (рис. 1.17).

Рис. 1.17. Отображение сущности и атрибутов

Очень важно дать атрибуту правильное имя: он должен именоваться в единственном числе и иметь четкое смысловое значение. Соблюдение этого правила позволяет частично решить проблему нормализации данных уже на этапе определения атрибутов. Например, создание в сущности *Сотрудник* атрибута *Телефоны Сотрудника* противоречит требованиям нормализации, так как атрибут должен быть атомарным, т.е. не содержащим множественных значений. Согласно синтаксису IDEF1X имя атрибута должно быть уникально в рамках модели (а не только в рамках сущности!); следовательно, новый атрибут, если его имя совпадает с уже существующим, должен быть переименован. На практике такое переименование не всегда удобно, поэтому по умолчанию эта опция выключена, однако в случае необходимости ее можно включить.

Для настройки правил уникальности имен модели (не только имен атрибутов) используют закладку Duplicate Names диалога Model Naming Options (меню Tools/Names/Model Naming Options) (рис. 1.18).

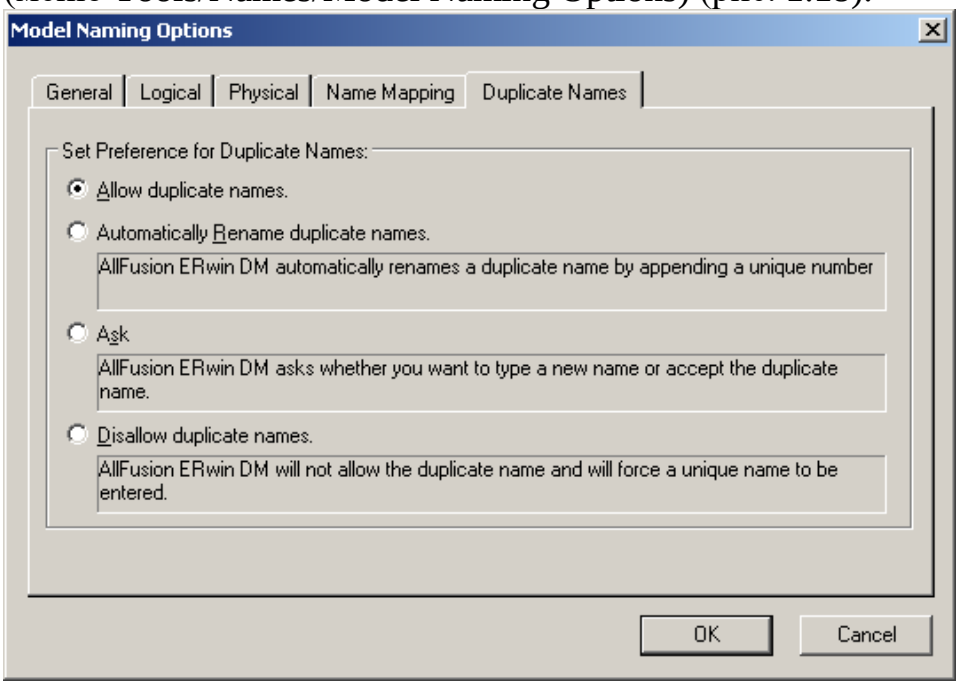


Рис. 1.18. Диалог Model Naming Options

Можно задать следующие режимы именования:

Allow duplicate name – позволить использование одинаковых имен (опция по умолчанию);

Automatically Rename duplicate names – автоматически переименовать атрибут (сущность) при попытке внесения уже существующего имени атрибута, при этом к имени атрибута добавляется число. Например, если атрибут *Имя* уже существует в модели, то при добавлении новых атрибутов *Имя* в ту же или в другую сущность они будут автоматически переименованы: *Имя_2*, затем *Имя_3* и т. д.;

Ask – запрашивать возможные действия каждый раз при внесении одноименных атрибутов (сущности). ERwin DM будет показывать на экране диалог Unique Name каждый раз, когда вводится неуникальное имя сущности или атрибута. В диалоге Unique Name можно ввести другое имя или разрешить дублирование (рис. 1.19). Новое имя уже не проверяется на уникальность;

Disallow duplicate names – запретить внесение одинаковых имен. При попытке использовать уже существующее имя ERwin DM сам производит переименование, чтобы обеспечить уникальность имен в модели.

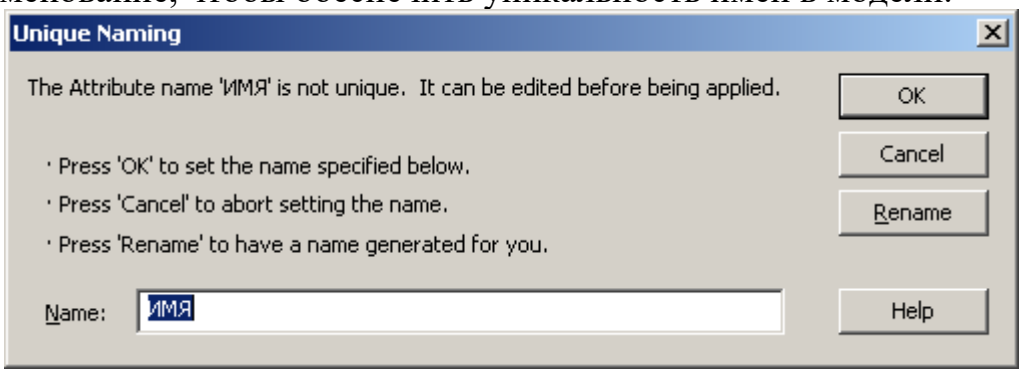


Рис. 1.19. Диалог Unique Name

Кроме общепринятых правил именования атрибутов часто требуется следовать правилам, разработанным внутри организации – корпоративным стандартам. Настроить правила именования можно с помощью диалогов *Model Naming Option* и *Edit Naming Standards* (меню Tools/Names).

Каждый атрибут должен быть определен в закладке Definition, при этом следует избегать циклических определений, например, когда термин 1 определяется через термин 2, термин 2 – через термин 3, а термин 3, в свою очередь, – через термин 1. Иногда легче дать определение атрибута через описание области значений. Например, *Оценка школьника* – это число, принимающее значения 2, 3, 4 или 5.

Часто приходится создавать *производные атрибуты*, значение которых можно вычислить из других атрибутов. Примером может служить *Возраст сотрудника*, который может быть вычислен из атрибута *Дата рождения сотрудника*. Такой атрибут может привести к конфликтам: если вовремя не обновить значение атрибута *Возраст сотрудника*, он может противоречить значению атрибута *Дата рождения сотрудника*. Производные атрибуты – ошибка нормализации, однако их вводят для повышения производительности системы – если необходимо узнать возраст сотрудника, можно обратиться к соответствующему атрибуту, а не проводить вычисления (которые на практике могут быть значительно более сложными, чем в приведенном примере).

ERwin DM позволяет перемещать атрибуты внутри сущности и между сущностями. Для этого необходимо щелкнуть левой кнопкой мыши по атрибуту. Указатель приобретает вид кисти руки, после чего можно, удерживая левую кнопку мыши, переместить атрибут.

Связи

Связь является логическим отношением между сущностями. Каждая связь должна именоваться глаголом или глагольной фразой (Relationship Verb Phrases) (рис. 1.20). Имя связи выражает некоторое ограничение или бизнес-правило и облегчает чтение диаграммы, например:

Каждый КЛИЕНТ <*размещает*> ЗАКАЗЫ;

Каждый ЗАКАЗ <*выполняется*> СОТРУДНИКОМ.

Рис. 1.20. Пример именования связей (Relationship Verb Phrases)

Связь показывает, какие именно заказы разместил клиент и какой именно сотрудник выполняет заказ. По умолчанию имя связи на диаграмме не показывается. Для отображения имени связи следует в контекстном меню, которое появляется, если щелкнуть правой кнопкой мыши по любому месту диаграммы, не занятому объектами модели, выбрать пункт Relationship Display и затем включить опцию Verb Phrase.

На логическом уровне можно установить идентифицирующую связь «один ко многим», связь «многие ко многим» и неидентифицирующую связь «один ко многим».

Связи идентифицирующие и неидентифицирующие

В IDEF1X и в IE различают *зависимые* и *независимые* сущности. Тип сущности определяется ее связью с другими сущностями. Идентифицирующая связь устанавливается между независимой (родительский конец связи) и зависимой (дочерний конец связи) сущностями. Когда рисуется *идентифицирующая* связь, ERwin DM автоматически преобразует дочернюю сущность в зависимую. Зависимая сущность изображается прямоугольником со скругленными углами (сущность *Заказ* на рис. 1.21). Экземпляр зависимой сущности определяется только через отношение к родительской сущности, т.е. в структуре информация о заказе не может быть внесена и не имеет смысла без информации о клиенте, который его размещает.

При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав первичного ключа дочерней сущности. Эта операция дополнения атрибутов дочерней сущности при создании связи называется *миграцией* атрибутов. В дочерней сущности новые атрибуты помечаются как

внешний ключ Foreign Key – *FK* (рис. 1.21). В дальнейшем, при генерации схемы базы данных, атрибуты первичного ключа получат признак NOT NULL, что означает невозможность внесения записи в таблицу заказов без информации о номере клиента.

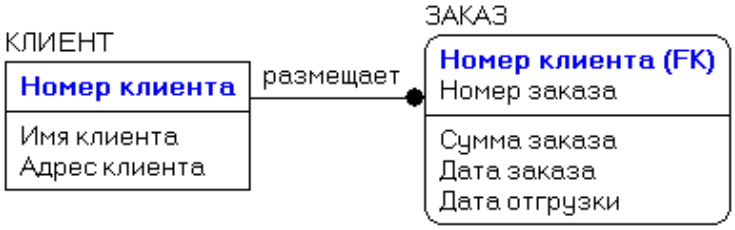


Рис. 1.21. Идентифицирующая связь

При установлении *неидентифицирующей* связи дочерняя сущность остается независимой, а атрибуты первичного ключа родительской сущности мигрируют в состав ее неключевых компонентов. Неидентифицирующая связь служит для соединения независимых сущностей. Экземпляр сущности *Сотрудник* может существовать безотносительно к какому-либо экземпляру сущности *Отдел*, т. е. сотрудник может работать в организации, не числясь в каком-либо отделе (рис. 1.22).

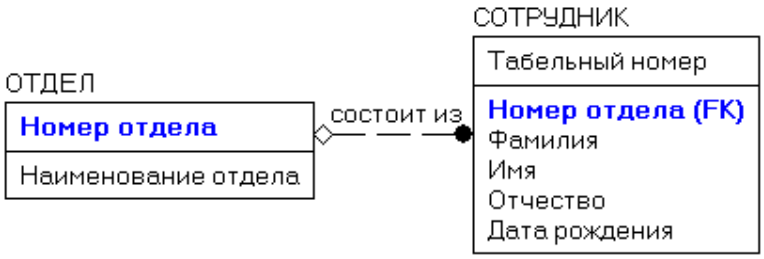


Рис. 1.22. Неидентифицирующая связь

Идентифицирующая связь показывается на диаграмме сплошной линией с жирной точкой на дочернем конце связи (см. рис. 1.21), неидентифицирующая – пунктирной (см. рис. 1.22).

Для создания новой связи:

щелкнуть левой кнопкой мышки по кнопке рисования связи в панели инструментов ERwin;

щелкнуть сначала по родительской, а затем по дочерней сущности.

Размещение фрагментов линии связи можно изменить: левой кнопкой мышки захватить нужный фрагмент линии связи и перенести его в другую позицию.

Для редактирования свойств связи щелкнуть правой кнопкой мыши по линии связи и выбрать в контекстном меню пункт Relationship Properties.

В появившемся диалоге Relationships в закладке *General* можно задать имя связи (раздел Verb Phrase), мощность (раздел Cardinality) и тип (раздел Relationship Type) (рис. 1.23).

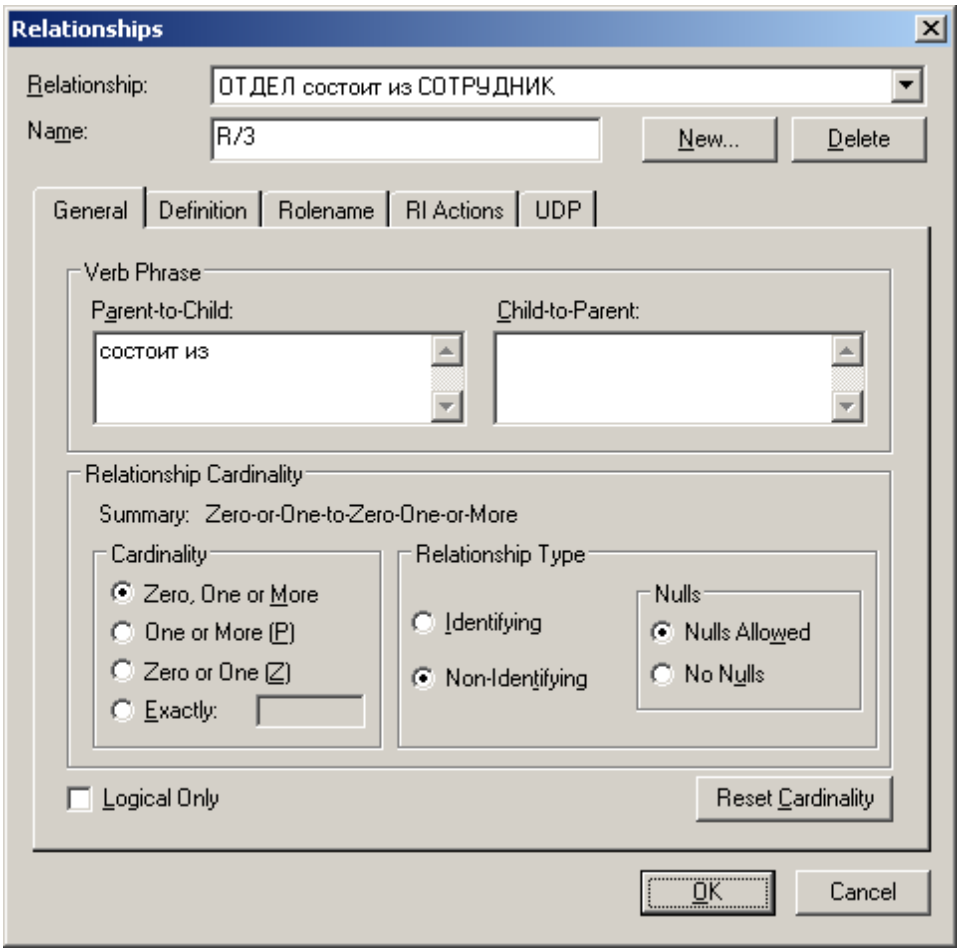


Рис. 1.23. Диалог Relationships

Имя связи (*Verb Phrase*) – фраза, характеризующая отношение между родительской и дочерней сущностями. Для связи «один ко многим» (идентифицирующей или неидентифицирующей) достаточно указать имя, характеризующее отношение от родительской к дочерней сущности (Parent-to-Child). Для связи «многие ко многим» следует указывать имя как Parent-to-Child, так и Child-to-Parent.

Мощность связи (*Cardinality*) служит для обозначения отношения числа экземпляров родительской сущности к числу экземпляров дочерней. Различают 4 типа мощности, которые были рассмотрены в разделе «Особенности методологий IDEF1X и IE». По умолчанию символ, обозначающий мощность связи, не показывается на диаграмме. Для его отображения следует в контекстном меню, которое появляется, если щелкнуть правой кнопкой мыши по любому месту диаграммы, не занятому объектами модели, выбрать пункт Relationship Display и затем включить опцию Cardinality.

Тип связи **(**Relationship Type**)**. Связи бывают идентифицирующими и неидентифицирующими. Для последних в разделе Nulls можно выбрать переключатель обязательности связи:

No Nulls – обязательная неидентифицирующая связь. При генерации схемы базы данных атрибут внешнего ключа получит признак NOT NULL, несмотря на то, что внешний ключ не войдет в состав первичного ключа дочерней сущности;

Nulls Allowed – необязательная неидентифицирующая связь, которая помечается прозрачным ромбом со стороны родительской сущности. Внешний ключ может принимать значение NULL.

В закладке *Definition* диалога Relationships можно дать более полное определение связи. В закладке *RI Actions* определяют правила ссылочной целостности (referential integrity), о которых будет рассказано позднее. Закладка *UDP* служит для задания значений свойств, определяемых пользователем. Предварительно эти свойства должны быть внесены в диалог User Defined Property (меню Model/UDP Dictionary) как свойства связи (Relationship). В закладке *Rolename* (рис. 1.24) можно задать имя роли.

Имя роли (*функциональное имя*) **–** показывает, какую роль играет мигрировавший атрибут в дочерней сущности. В примере, приведенном на рис. 1.25, в сущности *Сотрудник* внешний ключ *Номер отдела* имеет функциональное имя «*Где работает*», которое показывает, какую роль играет этот атрибут в сущности. По умолчанию в списке атрибутов показывается только имя роли.

Для отображения полного имени атрибута (имя роли + имя мигрировавшего атрибута) в контекстном меню, которое появляется, если щелкнуть правой кнопкой мыши по любому месту диаграммы, не занятому объектами модели, следует выбрать пункт Entity Display и затем включить опцию Rolename/Attribute. Полное имя показывается как имена роли и мигрировавшего атрибута, разделенные точкой (рис. 1.25).

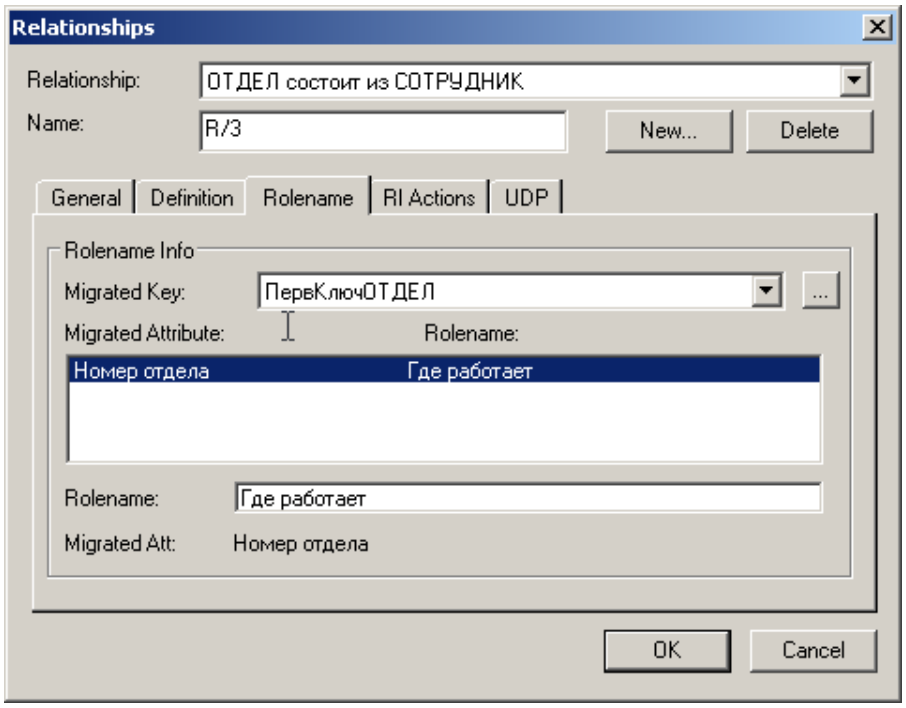


Рис. 1.24. Закладка Rolename диалога Relationships

Рис. 1.25. Пример имен ролей внешних ключей

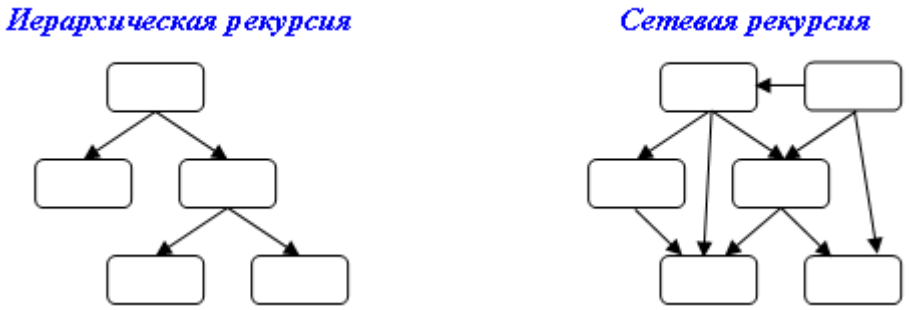
Обязательным является применение имен ролей в том случае, когда два или более атрибута одной и той же сущности имеют одинаковую область значений, но разный смысл. Сущность *Продажа валюты* содержит информацию об акте обмена валюты, в котором участвуют две валюты – проданная и купленная (рис. 1.26). Информация о валютах содержится в сущности *Валюта*. Следовательно, сущности *Продажа валюты* и *Валюта* должны быть связаны дважды, и первичный ключ *Номер валюты* – дважды мигрировать в сущность *Продажа валюты* в качестве внешнего ключа. Требуется различать два атрибута, мигрировавших из сущности *Валюта*: один содержит номер проданной валюты, а другой – номер купленной валюты, – они имеют общую область

значений и ссылаются на одну и ту же сущность. Атрибутам присвоены разные имена ролей: *Проданная* и *Купленная* (рис. 1.26).

Рис. 1.26. Пример обязательного использования имен ролей

Другим примером обязательности присвоения имен ролей являются рекурсивные связи, когда одна и та же сущность является родительской и дочерней одновременно. При задании рекурсивной связи атрибут должен мигрировать в качестве внешнего ключа в состав неключевых атрибутов той же сущности. Атрибут не может появиться дважды в одной сущности под одним именем, поэтому обязательно должен получить имя роли. Сущность *Сотрудник* содержит атрибут первичного ключа *Табельный номер*. Информация о руководителе сотрудника находится в той же сущности, поскольку руководитель работает в той же организации. Чтобы сослаться на руководителя сотрудника, следует создать рекурсивную связь (*Руководит/Подчиняется*) и присвоить имя роли (*Руководитель*). Заметим, что рекурсивная связь может быть только неидентифицирующей. В противном случае внешний ключ должен был бы войти в состав первичного ключа и получить при генерации схемы признак NOT NULL. Это сделало бы невозможным построение иерархии – у дерева подчиненности должен быть корень – сотрудник, который никому не подчиняется. Связь *Руководит/Подчиняется* позволяет хранить древовидную иерархию подчиненности сотрудников. Такой вид рекурсивной связи называется *иерархической рекурсией* (hierarchical recursion) и задает связь, когда руководитель (экземпляр родительской сущности) может иметь множество подчиненных (экземпляров дочерней сущности), но подчиненный имеет только одного руководителя (рис.1.27а).

Другим видом рекурсии является *сетевая рекурсия* (network recursion) (рис. 1.27б), когда руководитель может иметь множество подчиненных и, наоборот, подчиненный может иметь множество руководителей. Сетевая рекурсия задает паутину отношений между экземплярами родительской и дочерней сущностей. Это случай, когда сущность находится сама с собой в связи «многие ко многим». Для разрешения связи «многие ко многим» необходимо создать новую сущность (подробно связь «многие ко многим» будет рассмотрена ниже).



а б

Рис. 1.27. Подчиненность экземпляров сущности в рекурсии

Пример реализации сетевой рекурсии рассмотрен на рис. 1.28. Структура моделирует родственные отношения между членами семьи любой сложности. Атрибут *Тип отношения* может принимать значения «отец-сын», «мать-дочь», «дед-внук», «свекровь-невестка», «тесть-зять» и т. д. Поскольку родственное отношение связывает всегда двух людей, от сущности *Родственник* к сущности *Родственное отношение* установлены две идентифицирующие связи с именами ролей «Старший» и «Младший». Каждый член семьи может быть в родственных отношениях с любым другим членом семьи, более того, одну и ту же пару родственников могут связывать разные типы родственных отношений.

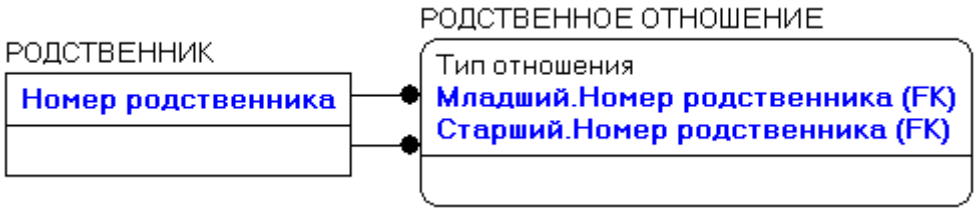


Рис. 1.28. Пример реализации сетевой рекурсии

Если атрибут мигрирует в качестве внешнего ключа более чем на один уровень, то на первом уровне отображается полное имя внешнего ключа (имя роли + базовое имя атрибута), на втором и более – только имя роли. Структура данных, содержащая сущность *Команда*, сущность *Игрок*, в которой хранится информация об игроках каждой команды, и сущность *Гол*, дающая информацию о голах, забитых каждым игроком, изображена на рис. 1.29. Атрибут внешнего ключа *Номер команды* сущности *Игрок* имеет имя роли *В какой команде играет*. На следующем уровне, в сущности *Гол*, отображается только имя роли соответствующего атрибута внешнего ключа (*В какой команде играет*).

Рис. 1.29. Пример миграции имен ролей

Правила ссылочной целостности (*RI – referential integrity*) определяют, что произойдет в случае, если будут произведены изменения в родительской или дочерней сущности: добавление (INSERT), обновление (UPDATE), удаление (DELETE). Позволяют избежать «висячих» данных и дают возможность соблюдать бизнес-правила.

ERwin DM автоматически присваивает каждой связи значение ссылочной целостности, устанавливаемой по умолчанию, прежде чем добавить ее в диаграмму. В закладке *RI Actions* диалога Model Properties (меню Model/Model Properties) можно изменить правила ссылочной целостности по умолчанию. На основе этих правил при генерации схемы базы данных будут сгенерированы правила декларативной ссылочной целостности для каждой связи и триггеры, обеспечивающие ссылочную целостность.

Переопределить правила ссылочной целостности для конкретной связи можно в закладке *RI Actions* диалога Relationships (рис. 1.30).

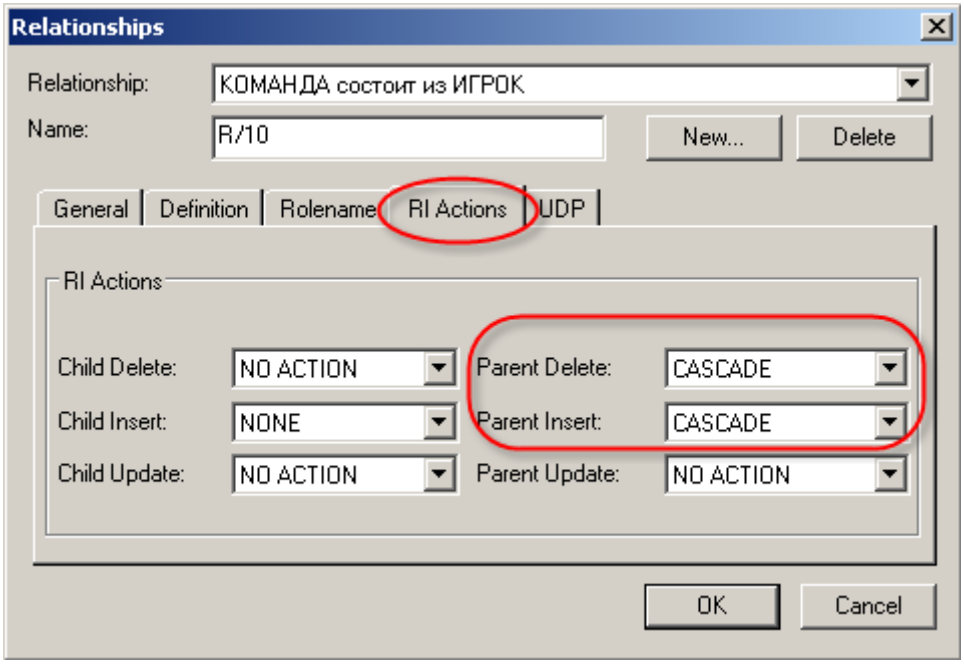


Рис. 1.30. Закладка RI Actions диалога Relationships

Триггеры представляют собой программы, которые срабатывают всякий раз при выполнении команд вставки, замены или удаления. По умолчанию ERwin DM генерирует триггеры, дублирующие декларативную ссылочную целостность.

Для связей в модели ERwin DM можно определить следующие типы действий триггеров ссылочной целостности (в зависимости от выбранной СУБД и ее версии этот список может быть короче):

Restrict. Не разрешает удалять, обновлять или редактировать экземпляр в родительской или дочерней сущности (таблице), если существует один или более связанных с ним экземпляров в дочерней или родительской сущности (таблице).

Cascade. Если экземпляр в родительской сущности (таблице) удаляется, вставляется или обновляется, то каждый связанный экземпляр в дочерней сущности (таблице) также удаляется, вставляется или обновляется.

Set null. Если экземпляр в родительской сущности (таблице) удаляется, вставляется или обновляется, то атрибут (колонка) внешнего ключа каждого связанного экземпляра дочерней сущности (таблице) устанавливается в NULL.

Set default. Если экземпляр в родительской сущности (таблице) удаляется, вставляется или обновляется, то атрибут (колонка) внешнего ключа каждому связанному экземпляру дочерней сущности (таблице) назначается определенное значение по умолчанию.

No Action. Если экземпляр в родительской сущности (таблице) удаляется, вставляется или обновляется, то во всех вязанных экземплярах дочерней сущности никакие действия не производятся.

None. Не требуются действия по поддержанию ссылочной целостности.

Для отображения на диаграмме установленных правил ссылочной целостности следует в меню Format выбрать пункт Relationship Display, затем Referential Integrity.

Примеры правил ссылочной целостности при удалении строки родительской таблицы приведены в табл. 1.2.

Ситуация, когда при удалении значение атрибута внешнего ключа не меняется (режим Parent Delete NONE) характерна для «плоских» таблиц. Например, если информация об игроках и командах хранится в *dbf*-файлах, можно удалить запись о команде, при этом файл игроков «ничего не будет знать» о том, что соответствующей команды не существует. Поэтому в настольных или файл-серверных системах функциональность, обеспечивающая правила ссылочной целостности, реализуется в клиентском приложении.

Правила удаления управляют тем, что будет происходить в базе данных при удалении строки, а *правила вставки и обновления* – тем, что

будет происходить с базой данных, если строки изменяются или добавляются. Например, можно установить правило, которое разрешает вносить новую команду только в том случае, когда в нее зачислен хотя бы один игрок. Желаемое поведение может быть достигнуто следующими действиями:

задать мощность связи между сущностями *Команда* и *Игрок*, равную 1 или более (тип P). Предполагается, что установлена идентифицирующая связь;

присвоить действие RI-триггера *Parent Insert-CASCADE* для того, чтобы при создании новой строки в таблице *Команда* автоматически создавалась хотя бы одна строка в дочерней таблице *Игрок*;

присвоить связи действие RI-триггера *Parent Delete-CASCADE*, чтобы при удалении строки из таблицы *Команда* соответствующая строка или строки из таблицы *Игрок* тоже удалялись (см. рис. 1.30).

Таблица 1.2. Правила ссылочной целостности при удалении строки родительской таблицы

№	Название	Пример
1	Parent Delete RESTRICT – удаление с ограничением	Между сущностями *Команда* и *Игрок* существует идентифицирующая связь (см. рис. 1.29). Экземпляр сущности *Игрок* не может существовать без *Команды* (атрибут первичного ключа В какой команде играет. Номер команды не может принимать значение NULL). Правило запрещает удаление команды, пока в ней числится хотя бы один игрок (для удаления команды сначала нужно удалить всех игроков). При попытке выполнить удаление строки из родительской таблицы *Команда*, в которой есть хотя бы один игрок, сервер реляционной СУБД возвратит ошибку
2	Parent Delete CASCADE – удаление каскадом	Между сущностями *Команда* и *Игрок* существует идентифицирующая связь (см. рис. 1.29). Экземпляр сущности *Игрок* не может существовать без команды (атрибут первичного ключа *В какой команде играет*. Номер команды не может принимать значение NULL). Согласно правилу вместе с командой удаляются сразу все ее игроки. Примечание. Сущности *Игрок* и *Гол*, в свою очередь, тоже связаны идентифицирующей связью и в случае удаления каскадом команды будут удалены все игроки этой команды и все голы, которые они забили. Выполнение команды на удаление одной строки фактически может привести к удалению тысяч строк в базе данных, поэтому использовать правило удаления каскадом следует с осторожностью
3	Parent Delete SET NULL – удаление с установкой в Null	Установлена необязательная неидентифицирующая связь между сущностями *Отдел* и *Сотрудник* (см. рис. 1.22). Экземпляр сущности *Сотрудник* может существовать без ссылки на отдел (атрибут внешнего ключа *Где работает*. Номер отдела может принимать значение NULL). Согласно правилу при удалении отдела атрибут внешнего

		ключа сущности *Сотрудник – Где работает*. Номер отдела примет значение NULL. Это означает, что при удалении отдела сотрудник остается работать в организации, не будучи приписан к какому-либо отделу, и информация о нем сохраняется
4	Parent Delete SET DEFAULT – удаление с установкой значений по умолчанию	Существует идентифицирующая связь между сущностями *Команда* и *Игрок* (см. рис. 1.29). Согласно правилу атрибут внешнего ключа получит значение по умолчанию., т. е. при удалении команды атрибут первичного ключа в сущности *Игрок* (*В какой команде играет. Номер команды*) получит значение по умолчанию. Например, при удалении команды ее игроки могут быть переведены в другую команду
5	Parent Delete NONE – простое удаление	Существует идентифицирующая связь между сущностями Команда и Игрок (см. рис. 1.29). Согласно правилу при удалении значение атрибута внешнего ключа не меняется. Запись об игроке ссылается на несуществующую уже команду

Связь «многие ко многим»

Связь «многие ко многим» может быть создана только на уровне логической модели. Пример связи «многие ко многим» показан на рис. 1.31. Врач может принимать много пациентов, пациент может лечиться у нескольких врачей. Такая связь обозначается сплошной линией с двумя точками на концах.

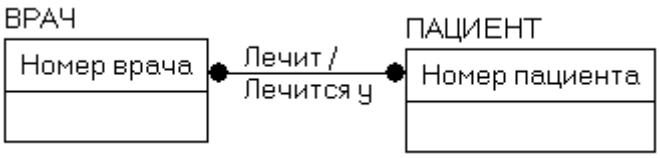


Рис. 1.31. Связь «многие ко многим»

Для внесения связи следует установить курсор на кнопке  на панели инструментов ERwin, щелкнуть по одной, затем по другой сущности.

Связь «многие ко многим» должна именоваться двумя фразами – в обе стороны (в примере «лечит/лечится у»). Это облегчает чтение диаграммы. Связь на рис. 1.31 следует читать так: *Врач <лечит> Пациента, Пациент <лечится у> Врача*.

На физическом уровне связь «многие ко многим» должна быть преобразована. По умолчанию при переходе к физическому уровню ERwin DM автоматически не преобразует связь «многие ко многим», и на физическом уровне диаграмма выглядит так же, как и на логическом. Однако при генерации схемы такая связь игнорируется.

Для преобразования связи «многие ко многим» необходимо щелкнуть по связи правой кнопкой мыши и выбрать пункт меню Create Association Table или выбрать связь и щелкнуть по инструменту  на

панели трансформаций ERwin Transform Toolbar (подробнее операции преобразования будут рассмотрены разделе «Трансформации»). Появится Мастер трансформаций – Many-To-Many Transform Wizard, состоящий из 4 шагов-диалогов. Для перехода к следующему шагу нужно щелкнуть по кнопке Next (Далее). На втором и третьем шаге следует задать имя вновь создаваемой таблицы и имя преобразования. Преобразование связи включает создание новой таблицы и двух новых связей «один ко многим» от старых к новой таблице (рис. 1.32). При этом имя новой таблице присваивается как *Имя1_Имя2*.

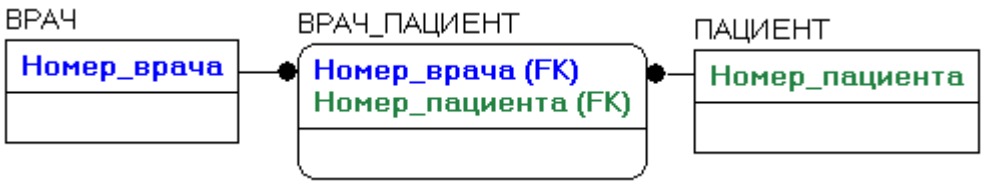


Рис. 1.32. Автотрансформация связи «многие ко многим»

Описанного выше решения проблемы связи «многие ко многим» не всегда оказывается достаточно. В примере таблица *Врач_Пациент* имеет смысл визита к врачу, поэтому согласно бизнес-логике ее следует переименовать в *Посещение*. Один и тот же пациент может много раз посещать врача, поэтому для того, чтобы идентифицировать визит, необходимо в состав первичного ключа таблицы *Посещение* добавить дополнительную колонку, например, дату и время посещения *(Дата_Время_Посещения*, рис. 1.33).

Следует заметить, что после переименования таблицы на физическом уровне представление модели на логическом уровне не изменится, диаграмма будет выглядеть так, как на рис. 1.32.

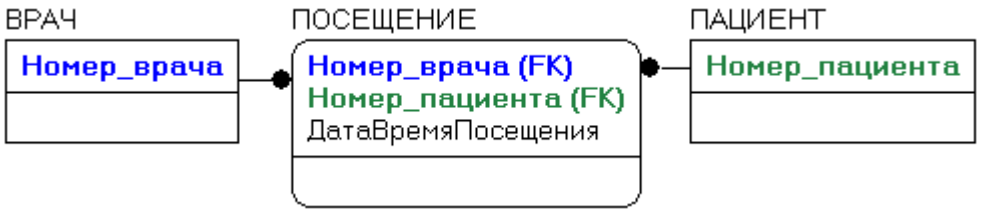


Рис. 1.33. Дополнение физического уровня модели после трансформации связи «многие ко многим»

Типы зависимых сущностей

Как было указано выше, связи определяют, является ли сущность независимой или зависимой. Различают несколько типов зависимых сущностей.

Характеристическая – зависимая дочерняя сущность, которая связана только с одной родительской и по смыслу хранит информацию о характеристиках родительской сущности (рис. 1.34).

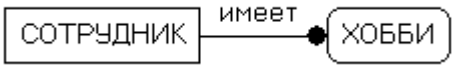


Рис. 1.34. Характеристическая сущность «Хобби»

Ассоциативная – сущность, связанная с несколькими родительскими сущностями. Содержит информацию о связях сущностей. В качестве примера можно привести *Посещение* (см. рис. 1.33).

Именующая – частный случай ассоциативной сущности, не имеющей собственных атрибутов (только атрибуты родительских сущностей, мигрировавших в качестве внешнего ключа). Примером является *Врач_Пациент* (см. рис. 1.32).

Категориальная – дочерняя сущность в иерархии наследования.

Иерархия категорий (иерархия наследования)

Представление об иерархиях категорий, их типах, отображении в нотациях IDEF1X и IE было дано в разделе «Особенности методологий IDEF1X и IE».

Рассмотрим возможные стадии построения иерархии наследования.

1. Определение сущностей с общими (но определению) атрибутами.

Предположим, что в процессе проектирования созданы сущности *Постоянный сотрудник* и *Совместитель* (рис. 1.35). Можно заметить, что часть атрибутов у этих сущностей (*Фамилия, Имя, Отчество, Дата рождения, Должность*) имеет одинаковый смысл.

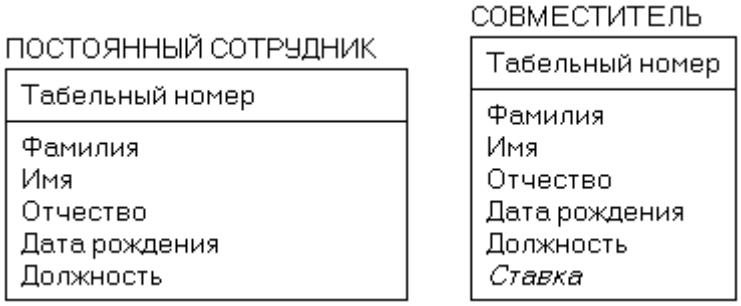


Рис. 1.35. Сущности с общими по смыслу атрибутами

В случае обнаружения совпадающих по смыслу атрибутов следует создать новую сущность (*Сотрудник*) – родовой предок и перенести в нее общие атрибуты.

2. Создание неполной структуры категорий. Создается категориальная связь от новой сущности – родового предка к старым

сущностям-потомкам. Новая сущность дополняется атрибутом-дискриминатором категории (*Тип*) (рис. 1.36).

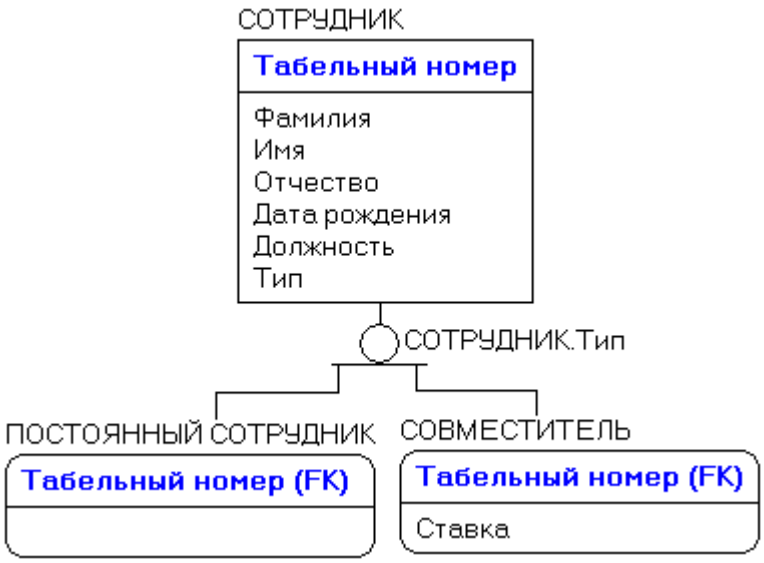


Рис. 1.36. Пример неполной иерархии категорий

Для создания категориальной связи:

щелкнуть левой кнопкой мыши по кнопке ;

щелкнуть сначала по родовому предку, а затем по потомку;

для установления второй связи в иерархии категории сначала щелкнуть по символу категории, затем по второму (третьему и т. д.) потомку.

Для редактирования категорий нужно щелкнуть правой кнопкой мыши по символу категории и выбрать в контекстном меню пункт Subtype Properties. В диалоге Subtype Properties (рис. 1.37) можно указать атрибут-дискриминатор категории *Тип* (список Discriminator) и тип категории – *Incomplete* – неполная (раздел Type: опции Complete/Incomplete – полная/неполная).

1. Создание полной структуры категорий. Проводится дополнительный поиск сущностей, имеющих общие по смыслу атрибуты с родовым предком. В примере это сущность *Консультант*.

Общие атрибуты переносятся в родового предка, и категория преобразуется в полную. Признак полной категории устанавливается в диалоге Subtype Relationship (в разделе Type следует выбрать опцию *Complete*).

Сущность *Консультант* не имеет атрибута *Должность*, поэтому в родовом предке значение этого атрибута в случае консультанта будет NULL. В зависимости от бизнес-правил атрибут *Должность* переносится обратно из родового предка в сущности-потомки *Постоянный сотрудник* и *Совместитель* или принимается решение о том, что для консультанта также требуется указывать должность (рис. 1.38).

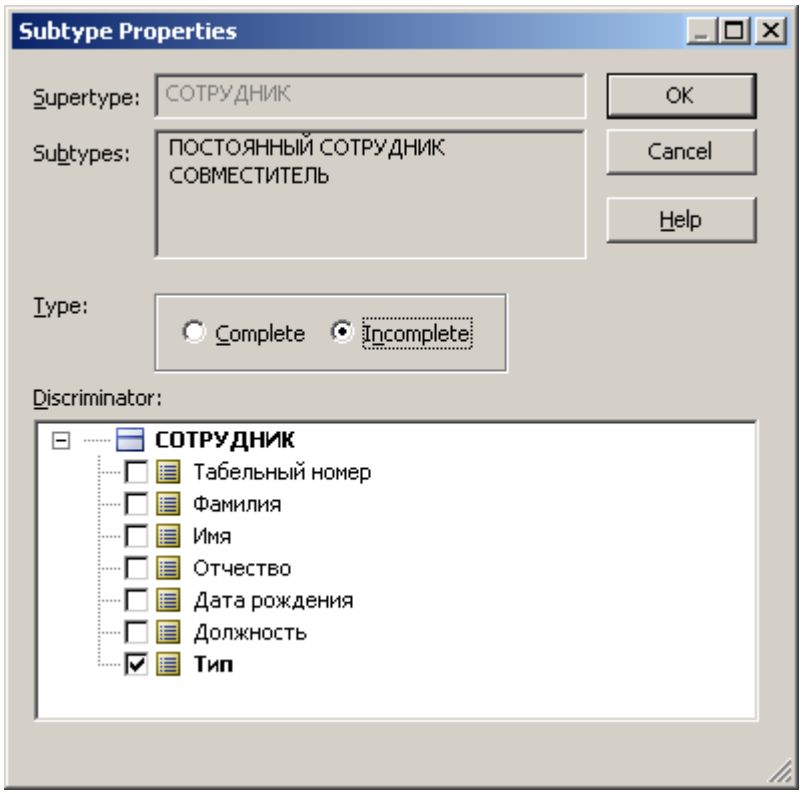


Рис. 1.37. Диалог Subtype Properties

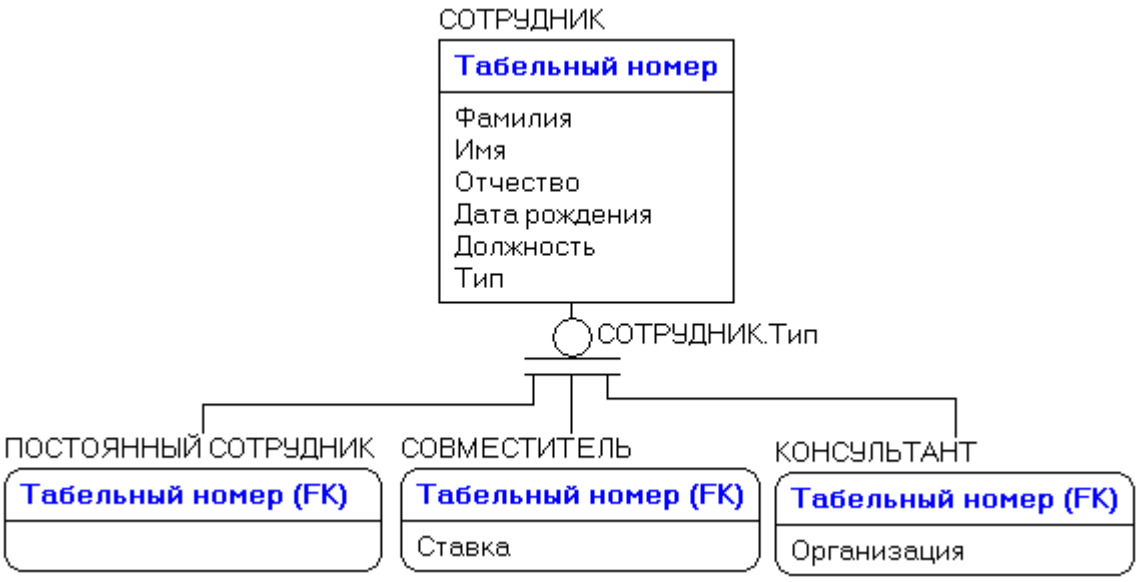


Рис. 1.38. Полная иерархия категорий

2. Пример комбинации полной и неполной категорий показан на рис. 1.39. Согласно этому фрагменту модели сотрудник может быть совместителем или работать постоянно (неполная категория, так как не

отображены сотрудники-консультанты), а постоянный сотрудник является либо мужчиной, либо женщиной (полная категория).

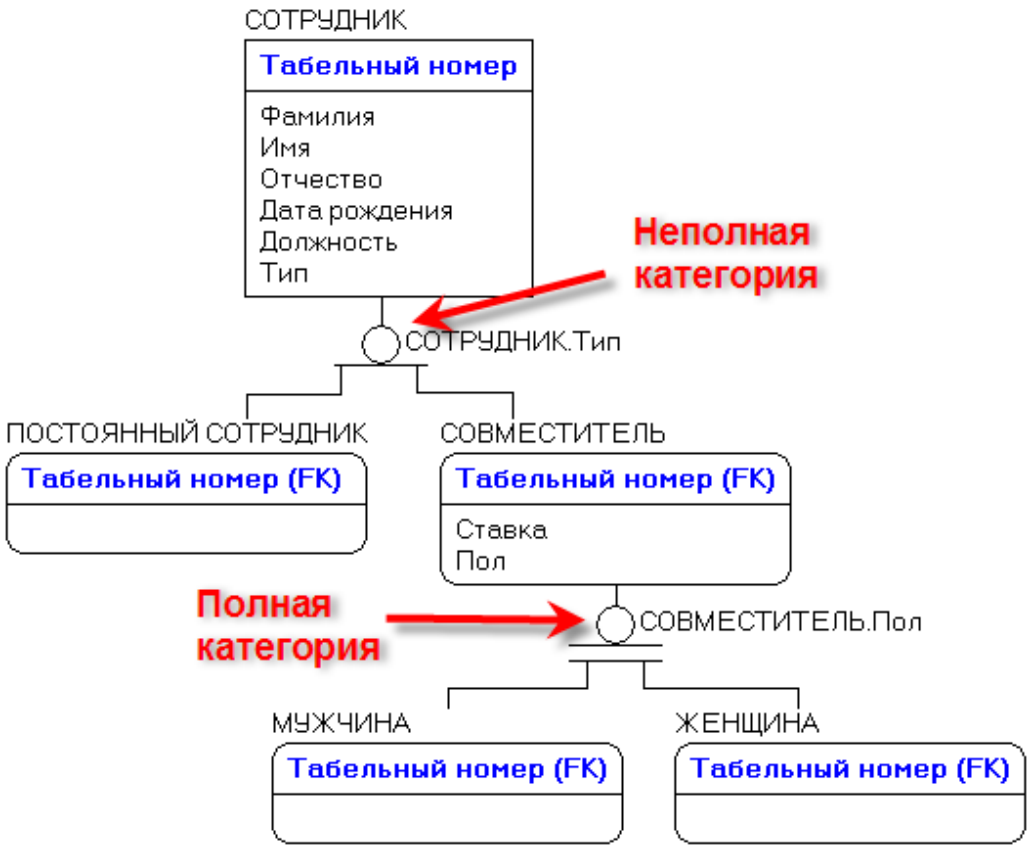


Рис. 1.39. Смешанная иерархия категорий

Ключи

Выделяют потенциальные, первичные, сложные, альтернативные, инверсные, внешние, суррогатные ключи. Каждый экземпляр сущности должен быть уникален.

Первичный ключ (*primary key*) – это атрибут или группа атрибутов, уникально идентифицирующая каждый экземпляр сущности. Атрибуты первичного ключа на диаграмме располагаются выше горизонтальной линии (рис. 6.40). При внесении нового атрибута в диалоге Attributes для того, чтобы сделать его атрибутом первичного ключа, нужно включить флажок Primary Key в нижней части закладки General. На диаграмме неключевой атрибут можно перевести в состав первичного ключа, воспользовавшись режимом переноса атрибутов.

Выбор первичного ключа может оказаться непростой задачей, решение которой в состоянии повлиять на эффективность будущей информационной системы. В одной сущности могут оказаться несколько атрибутов или их наборов, претендующих на роль первичного ключа.

Такие претенденты называются *потенциальными ключами* (*candidate key*).

Ключи могут быть *сложными* (*составными*), т. е. содержащими несколько атрибутов.

Рис. 1.40. Ключи сущности «Сотрудник»

Рассмотрим потенциальные ключи сущности *Сотрудник* (см. рис. 1.40):

1. Табельный номер.
2. Номер паспорта.
3. Фамилия + Имя + Отчество.

Выберем из них первичный ключ. Требования, которым он должен соответствовать: уникальность, компактность, простота. Кроме того, этот ключ не должен содержать нулевых (отсутствующих) значений, а значение атрибутов ключа не должно меняться в течение всего времени существования экземпляра сущности. Рассмотрим эти требования подробнее.

Уникальность. Два экземпляра не должны иметь одинаковых значений возможного ключа. Потенциальный ключ № 3 (*Фамилия* + *Имя* + *Отчество*) является плохим кандидатом, поскольку в организации могут работать полные тезки, поэтому добавим атрибут дату рождения: *Фамилия* + *Имя* + *Отчество* + *Дата рождения*.

Компактность. Сложный потенциальный ключ не должен содержать ни одного атрибута, удаление которого приводило бы к утрате уникальности. Для обеспечения уникальности ключа № 3 дополним его атрибутами *Дата рождения* и *Цвет волос.* Если бизнес-правила говорят, что сочетания атрибутов *Фамилия* + *Имя* + *Отчество* + *Дата рождения* достаточно для однозначной идентификации сотрудника, то *Цвет волос* оказывается лишним, т. е. ключ *Фамилия* + *Имя* + *Отчество* + *Дата рождения* + *Цвет волос* не является компактным.

При выборе первичного ключа предпочтение должно отдаваться более простым ключам, содержащим меньшее количество атрибутов. В рассматриваемом примере потенциальные ключи № 1 и № 2 предпочтительнее ключа № 3.

Атрибуты ключа не должны содержать нулевых значений. Если допускается, что сотрудник может не иметь паспорта или вместо паспорта у него какое-либо другое удостоверение личности, то потенциальный ключ № 2 не подойдет на роль первичного. Если для обеспечения уникальности необходимо дополнить потенциальный ключ второстепенными атрибутами, то они не должны содержать нулевых значений. Дополняя ключ № 3 атрибутом *Дата рождения*, нужно убедиться в том, что даты рождения известны для всех сотрудников.

Значение атрибутов ключа не должно меняться в течение всего времени существования экземпляра сущности. Сотрудница организации может выйти замуж и сменить как фамилию, так и паспорт. Поэтому ключи № 2 и 3 не подходят на роль первичного ключа.

Иногда создают *искусственный* (*суррогатный*) ключ, например, *Номер сотрудника, Номер клиента, Номер товара* и т. д.

Каждая сущность должна иметь по крайней мере один потенциальный ключ. У многих сущностей он только один и поэтому становится первичным. Если же в сущности более одного возможного ключа, то один из них становится первичным, а остальные – альтернативными.

Альтернативный ключ (*Alternate Key*) – потенциальный ключ, не ставший первичным. ERwin DM позволяет выделить атрибуты альтернативных ключей, и по умолчанию при генерации схемы базы данных по этим атрибутам в дальнейшем будет генерироваться уникальный индекс.

Инверсный вход (*Inversion Entry*) – атрибут или группа атрибутов, которые не определяют экземпляр сущности уникальным образом, но часто используются в запросах к базе данных для обеспечения доступа к нескольким экземплярам сущности, объединенным каким-либо признаком. В этом случае для повышения производительности информационной системы используются неуникальные индексы. ERwin DM позволяет на уровне логической модели назначить атрибуты, которые будут участвовать в неуникальных индексах, а затем сгенерировать неуникальный индекс для каждого Inversion Entry.

В ERwin DM создать альтернативные ключи и инверсионные входы можно в диалоге Key Groups (рис. 1.41). Для запуска диалога следует в меню Model выбрать пункт Key Groups или щелкнуть правой кнопкой мыши по сущности и в появившемся контекстном меню выбрать пункт Key Groups. В верхней части диалога находится список сущностей, в средней – список ключей, в нижней – список атрибутов, доступных для

включения в состав ключа (слева), и список выбранных ключевых атрибутов (справа). Каждый вновь созданный ключ должен иметь хотя бы один атрибут. Каждому ключу соответствует индекс, имя которого также присваивается автоматически. Имена ключа и индекса можно изменить вручную.

На диаграмме атрибуты альтернативных ключей обозначаются как (AKn.m), где n – порядковый номер ключа, m – порядковый номер атрибута в ключе. Когда альтернативный ключ содержит несколько атрибутов, (AKn.m) ставится после каждого.

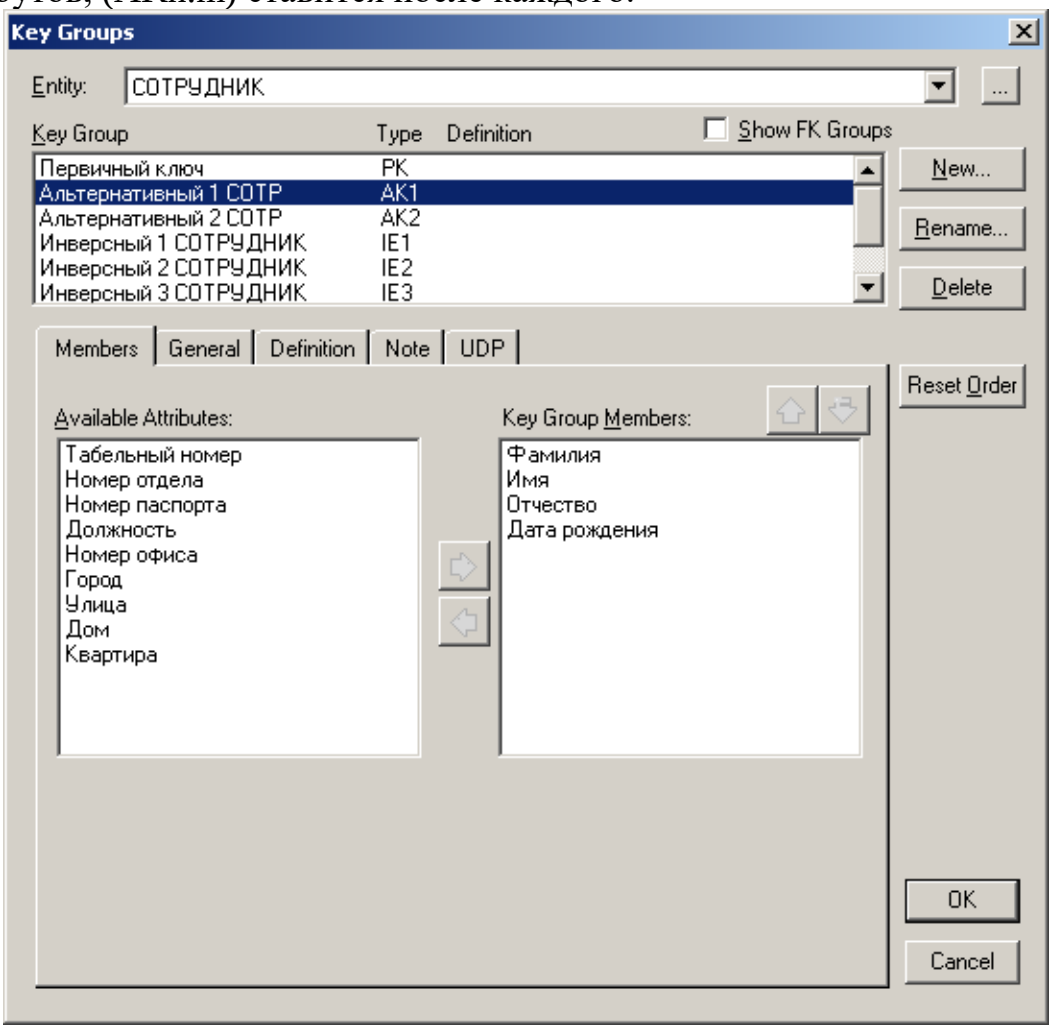


Рис. 1.41. Диалог Key Groups

Атрибуты *Фамилия, Имя, Отчество* и *Дата рождения* входят в альтернативный ключ № 1 (АК1), *Номер паспорта* составляет альтернативный ключ № 2 (АК2) (см. рис. 1.40). Инверсионные входы обозначаются как (IEn.m), где n – порядковый номер входа, m -порядковый номер атрибута. Инверсионный вход IE1 (атрибут *Должность*) позволяет выбрать всех сотрудников, занимающих одинаковую должность, IE2 (атрибут *Номер офиса*) – всех сотрудников, работающих в одном офисе, IE3 (атрибуты *Город* и *Улица*) – всех сотрудников, живущих на одной ули-

це. Если один атрибут входит в состав нескольких ключей, ключи перечисляются в скобках через запятую.

По умолчанию номера альтернативных ключей и инверсионных входов рядом с именем атрибута на диаграмме не показываются. Для отображения номера в контекстном меню, которое появляется, если щелкнуть правой кнопкой мыши по любому месту диаграммы, не занятому объектами модели, следует выбрать пункт Entity Display, затем включить опцию Alternate Key Designator (AK).

Внешние ключи (*Foreign Key*) создаются автоматически, когда связь соединяет сущности: связь образует ссылку на атрибуты первичного ключа родительской сущности, и эти атрибуты образуют внешний ключ в дочерней сущности (миграция атрибутов ключа). Атрибуты внешнего ключа обозначаются символом (FK) после своего имени. Атрибут внешнего ключа *Номер отдела* в сущности *Сотрудник* является атрибутом первичного ключа в сущности *Отдел***.**

Зависимая сущность может иметь один и тот же внешний ключ из нескольких родительских сущностей, а сущность – получить один и тот же внешний ключ несколько раз от одного и того же родителя через несколько разных связей. Когда ERwin DM обнаруживает одно из этих событий, он распознает, что два атрибута одинаковы, и помещает атрибут внешнего ключа в зависимую сущность только один раз. Хотя в закладке Key Group диалога Attribute этот атрибут будет входить в два внешних ключа, на диаграмме он показывается только один раз. Это комбинирование или объединение идентичных атрибутов называется *унификацией*. Она производится, поскольку правила нормализации запрещают существование в одной сущности двух атрибутов с одинаковыми именами. Однако есть случаи, когда унификация нежелательна. Например, если два атрибута имеют одинаковые имена, но на самом деле отличаются по смыслу, и необходимо, чтобы это отличие отражалось в диаграмме. Поэтому необходимо использовать имена ролей атрибутов внешнего ключа.

В некоторых случаях бывает целесообразно иметь в дочерней сущности ссылку не на первичный, а на альтернативный ключ. ERwin DM позволяет создавать связи, при которых в дочернюю сущность мигрируют атрибуты одного из альтернативных ключей. Для создания такой связи необходимо создать идентифицирующую или неидентифицирующую связь, щелкнуть по связи правой кнопкой мыши, выбрать пункт меню Relationship Properties, в открывшемся диалоге Relationships перейти на закладку Rolename и в выпадающем списке Migrated Key выбрать ключ, атрибуты которого будут мигрировать в дочернюю сущность. В результате в дочерней сущности внешний ключ будет содержать атрибуты альтернативного ключа родительской сущности (рис. 1.42).

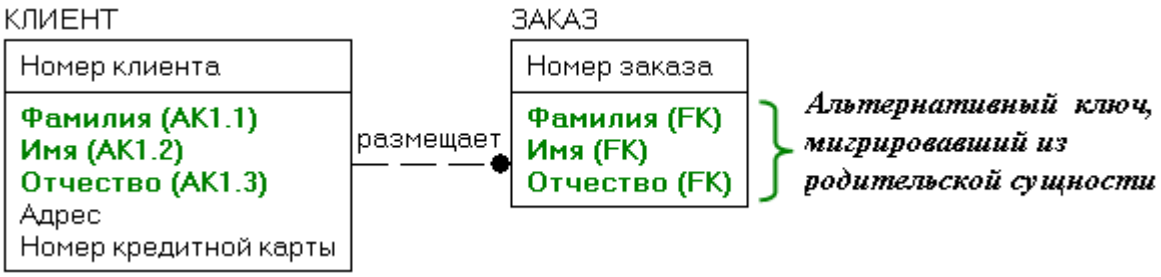


Рис. 1.42. Миграция атрибутов альтернативного ключа

1.5. Характеристика Power Designer

Power Designer – полнофункциональный инструментарий для создания бизнес-приложений, включающий в себя средства моделирования бизнес-процессов, возможности концептуального и физического проектирования баз данных, моделирования с использованием UML, и предоставляющий собой централизованный репозитарий для хранения моделей и объектов.

*Основные особенности продукта Power Designer***:**

моделирование бизнес-процессов на основе диаграмм потоков управления;

технологии моделирования данных (концептуальная и физическая модель), основанные на индустриальном стандарте «сущность/связь» (entity/relationship), включая технологии моделирования хранилищ данных (схемы «звезда» и «снежинка», многомерное моделирование, привязка к конкретному источнику данных);

стандартные диаграммы UML: use case, activity, sequence, диаграммы классов, компонентов и др.;

поддержка UML 2;

генерация на основе диаграмм классов исходных текстов для Java, C#, C++, PowerBuilder и VB.Net;

генерация операторов DDL (Data Definition Language) более чем для 50 РСУБД включая Oracle вплоть до версии 10g, IBM DB/2 до v8, Microsoft SQL Server 2000, Sybase ASE, ASA and IQ, MySQL и для многих других;

поддержка EJB 2.0;

определение сложных пользовательских типов данных, включая Java-классы и хранимые Java-процедуры, содержащиеся в БД;

обратное проектирование схемы базы данных в концептуальную и физическую модель;

обратное проектирование существующей бизнес-логики в диаграммы классов (Java, PowerBuilder, C#, VB.Net);

прямое и обратное проектирование XML-приложений в диаграммы классов. Поддержка XML-DTD, XML-схемы и XML-данных;

интеграция с популярными средствами разработки на Java и с ведущими сертифицированными под J2EE/EJB 2.0 серверами приложений;

requirement Model – специализированная модель для документирования и анализа требований предъявляемых к создаваемой информационной системе;

information Liquidity Model – модель, предназначенная для проектирования систем репликации данных;

Современный, графический, настраиваемый пользовательский интерфейс, содержащий общую оболочку, обозреватель объектов, область редактирования диаграмм и область состояния;

улучшенное управление моделями, включающее синхронизацию объектов, моделей и баз данных;

расширенный, не зависимый от модели генератор отчетов, который позволяет получить документ, включающий в себя информацию по нескольким моделям.

Централизованный репозитарий обеспечивает:

возможность одновременной работы над одной моделью большого числа различных аналитиков и проектировщиков;

хранение, управление и создание версий моделей PowerDesigner и других документов;

поиск объектов в модели и их повторное использование в других моделях;

эффективное управление взаимосвязями между моделями.

Преимущества Power Designer

Снижение затрат при разработке Web-служб через проектирование;

быстрая разработка с применением UML – с интеграцией со средствами разработки для ускорения создания Web-служб – делает простым и доступным создание сложных Web-служб;

быстрая разработка базы данных, которая поддерживает Web-службы в области хранения существующих или новых данных, упрощает взаимодействие разработчика и администратора базы данных;

оперативный учет всех изменений на этапе проектирования позволяет снизить общие затраты на разработку;

уникальная усовершенствованная технология синхронизации многочисленных моделей.

Поскольку бизнес-моделирование и техническое моделирование тесно взаимосвязаны, информация обо всех изменениях, происходящих в деловой сфере или на рынке, доводится непосредственно до сотрудников IT-отдела компании, позволяя им адаптировать Web-службы и дизайн системы в соответствии с требованиями бизнеса.

Основные возможности PowerDesigner включают в себя:

1. Моделирование бизнес-процессов. PowerDesigner позволяет бизнес-пользователям, не обладающим техническим опытом, но знающим

предметную область, проектировать бизнес-процессы в рамках удобной графической модели. Поддерживается генерация кода и его обращенное проектирование в рамках технологии ebXML.

2. Моделирование данных – проектирование и генерация структур базы данных путем концептуального и физического моделирования. Поддержка более 45 широкораспространенных на рынке баз данных, включая Oracle, Sybase ASE/ASA/IQ, IBM DB/2, Microsoft SQL Server и др. Возможность как генерации структуры БД по модели, так и восстановление модели по существующей структуре (reverse engineering). Также в PowerDesigner есть поддержка проектирования хранилищ данных.

3. Объектное моделирование приложений – полная поддержка анализа и проектирования в рамках методологии UML. Возможность работы с различными типами UML-диаграмм – диаграммы прецедентов (use case), активности (activity), последовательностей (sequence), классов (class), компонентов (component), взаимодействия (collaboration), состояния (statechart), объектов (object), развертывания (deployment). На базе диаграммы классов существует возможность генерации программного кода для таких языков как Java (включая EJB 2.0), XML, Web Services, C++/C#, PowerBuilder, Visual Basic.

Физические модели данных

Импорт моделей ERWin. Логические и физические модели ERWin могут быть одновременно импортированы в CDM, PDM и LDM модели Power Designer с полным сохранением метаданных и связей между моделями. Power Designer автоматически создает зависимости для использованных в хранимой процедуре таблиц, представлений (view), других процедур. Если хранимая процедура отображена на диаграмме (имеет собственный графический символ), эти зависимости могут быть графически отображены стрелками от объекта к объекту, что облегчает визуальный анализ зависимости кода и объектов БД друг от друга.

Объектно-ориентированные модели (*UML*)

PowerDesigner 12.0 обеспечивает поддержку спецификаций UML 2. Диаграммы классов и компонентов теперь поддерживают наличие портов и составных частей (parts). Символ компонента может отображать на диаграмме внутреннюю структуру его составных объектов. Новая составная структурная диаграмма позволяет представить символ объекта в виде составной вложенной диаграммы, показывающей внутреннее устройство использованных объектов и их взаимодействие с внешней средой. Диаграмма последовательностей поддерживает использование подфрагментов диаграммы взаимодействия и ссылки на внешние диаграммы взаимодействий.

Модели бизнес-процессов

Введена поддержка нотации BPMN 1.0, обеспечивающая высокоуровневое проектирование бизнес-процессов для их последующей

миграции/выполнения на физическом уровень в виде BPEL4WS или других исполняемых форматов бизнес-процессов. Появилась возможность импорта/экспорта структур данных бизнес-процесса в/из таблиц физических моделей данных.

1.6. Краткая характеристика Rational Rose

Альтернативой структурному подходу стали лишенные его недостатков объектно-ориентированные методы разработки информационных систем. Снижение риска в объектной технологии достигается за счет реализации технологии итерационной разработки (так называемая спиральная модель жизненного цикла). Разработка состоит из ряда итераций, которые в дальнейшем приводят к созданию информационной системы.

Модель представляет собой совокупность диаграмм, описывающих различные аспекты структуры и поведения информационной системы. Для просмотра модели в Rational Rose используется иерархический навигатор модели– Browser (рис. 1.43). В дальнейшем будет описан интерфейс версии Rational Rose for Java (version 4.0).

Рассмотрим в общих чертах некоторые диаграммы UML

Диаграммы использования системы (*Use Cases*) показывают, какая функциональность должна быть реализована в системе, основные функции, которые необходимо включить в систему (use case), их окружение (actors) и взаимодействие функций с окружением (рис. 1.44.). Воздействующие объекты (actors) не являются частью системы: это конечные пользователи или другие программы, взаимодействующие с проектируемой информационной системой.

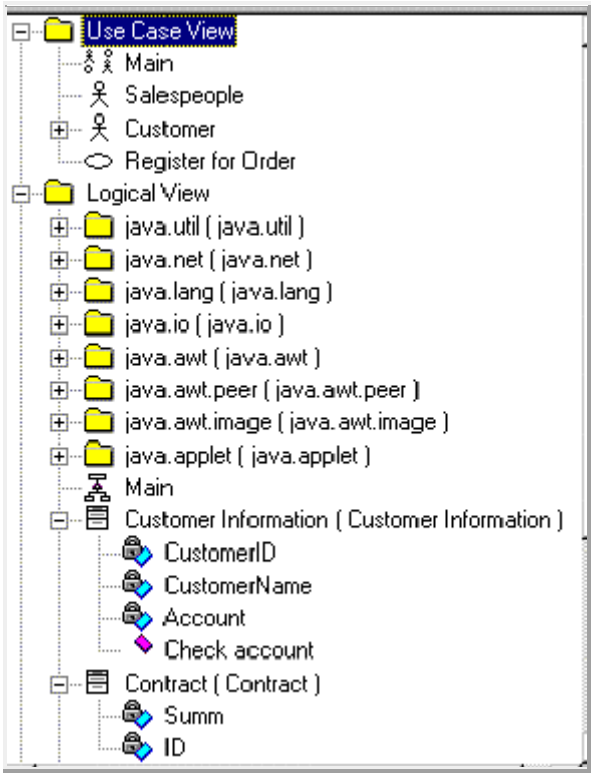


Рис. 1.43. Иерархический навигатор модели в Rational Rose

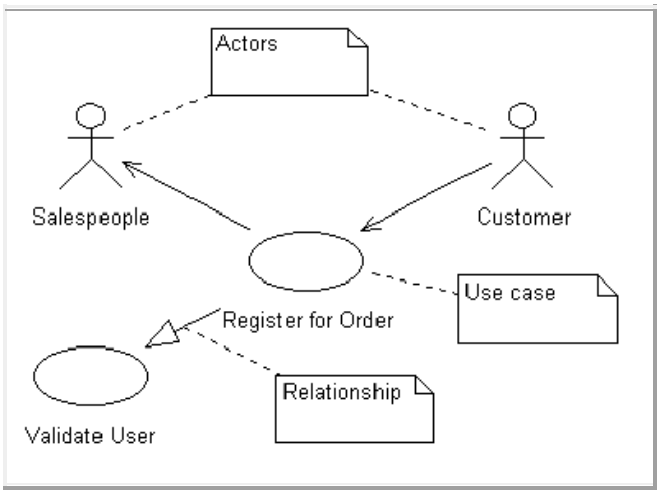


Рис. 1.44. Диаграмма Use Cases. Здесь Customer, Salespeople – Actors; Register for Order, Validate User – Use case

Диаграммы Use Cases включают отношения и ассоциации, показывающие взаимодействие между воздействующими объектами и функциями (изображ-ются в виде стрелок) и примечания (note), которые могут быть привязаны к любому объекту диаграммы Use Cases. Для создания новой диаграммы Use Cases следует правой кнопкой мыши кликнуть в навигаторе модели по закладке Use Case View и выбрать во всплывающем меню пункты New/Use Case. Для внесения в диаграмму Use

Case и установления связей между ними следует использовать кнопки палитры инструментов Rational Rose.

Диаграммы классов (рис. 1.45)**.** Под объектом в UML понимается некоторое абстрактное представление конкретного предмета, оторый имеет состояние, поведение и индивидуальность. Например, объект «Проект» может иметь два состояния – «открыт» и «закрыт». Поведение объекта определяет, как он взаимодействует с другими объектами. Индивидуальность означает, что каждый объект уникален. Под классом понимается описание объектов, обладающих общими свойствами (атрибутами), поведением, общими взаимоотношениями с другими объектами и общей семантикой. Класс является шаблоном для создания новых объектов. Для внесения нового класса в диаграмму классов нужно использовать кнопку  в палитре инструментов.

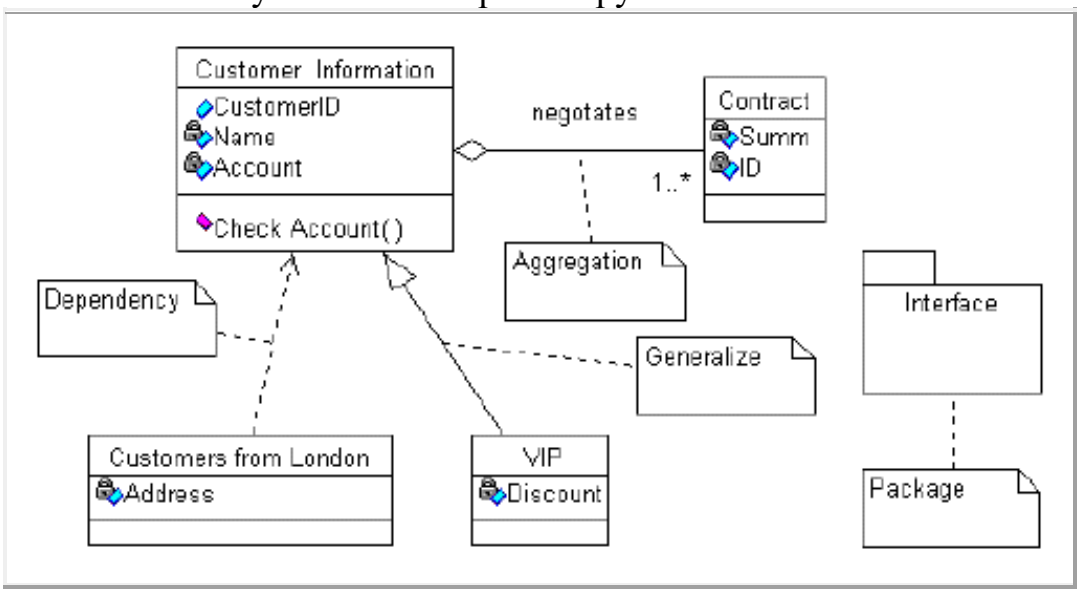


Рис. 1.45. Диаграмма классов

Если система содержит большое количество классов, они могут быть объединены в пакеты (package). Каждый класс может иметь атрибуты. Так, на рис. 1.45 класс Customer Information (информация о клиенте) имеет атрибуты CustomerID (идентификатор клиента), Name (имя) и Account (счет). Кроме того, каждый класс может методы (operations) – некоторые действия, которые описывают поведение объектов класса. На рис. 1.44 класс Customer Information имеет метод Check Account. Для внесения свойств класса следует правой кнопкой мыши кликнуть по классу и выбрать во всплывающем меню пункт Specification. Классы могут иметь взаимосвязи (relationship), называемые *отношениями.* В нотации UML имеется несколько типов отношений. Отношение использования (associations, кнопка  палитры инструментов) показывает, что объект одного класса связан с одним или несколькими объектами другого класса.

Отношение включения (aggregation, кнопка ) является частным случаем отношения использования. Оно показывает, что один объект является частью другого. При воздействии на один объект, связанный отношением включения, некоторые операции автоматически могут затронуть другой объект. Например, класс Customer Information связан отношением включения с классом Contract. При удалении объекта класса Customer Information (информация о клиенте) должны удаляться все объекты класса Contract (относящиеся к данному клиенту контракты). Каждая связь может быть охарактеризована определенной фразой, называемой именем роли. Связь между классами Customer Information и Contract имеет имя negotiates. Каждая связь может иметь индикатор множественности, который показывает, сколько объектов одного класса соответствует объекту другого класса. Связь negotiates имеет индикатор 1…* (один или много).

Наследование (inheritance) описывает взаимосвязь между классами, когда один класс (называется подклассом, subclass) получает структуру и/или поведение одного или нескольких классов. Подкласс VIP наследует свойства и поведение класса Customer Information. Связь классов в иерархии наследования называется отношением наследования (generalization, кнопка ).

Временная диаграмма (*Sequences*) демонстрирует поведение объектов во времени (рис. 1.46). Она показывает объекты и последовательность сообщений, посылаемых этими объектами. Сообщения на диаграмме сценариев изображаются в виде стрелок.

Архитектура приложения объясняется в диаграммах компонент (Component Diagram), которые описывают вхождение классов и объектов в программные компоненты системы (модули, библиотеки и т. д.) и диаграммы развертывания (Deployment Diagram), при помощи которых документируется размещение программных модулей на узлах (физических и логических устройствах) системы. В данной статье эти диаграммы не рассматриваются.

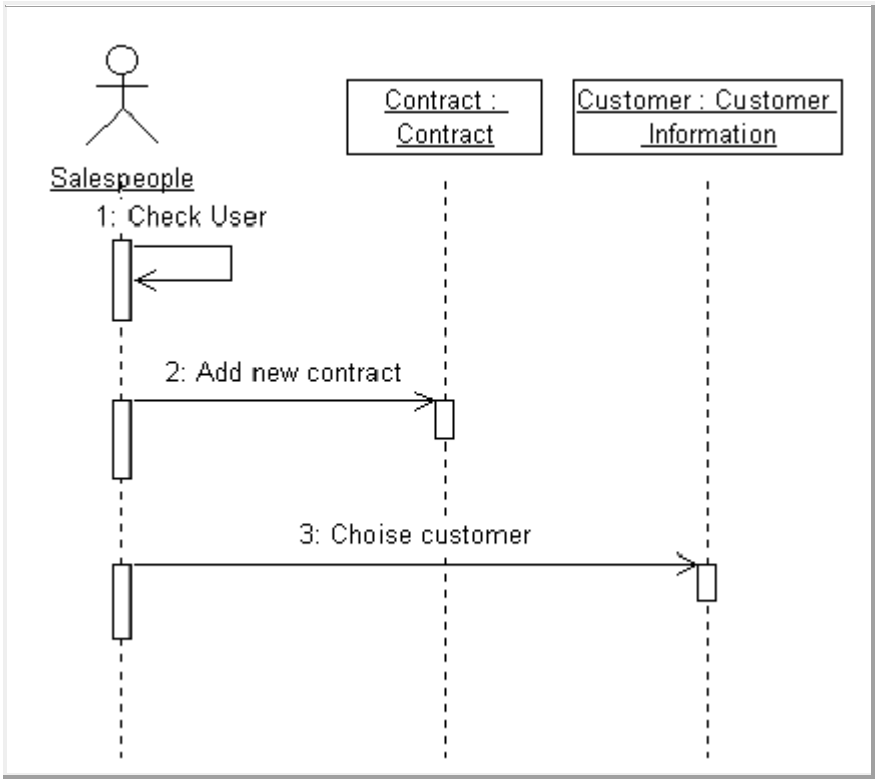


Рис. 1.46. Временная (*Sequence*) диаграмма

Генерация кода осуществляется на основе диаграмм классов. Необходимо выбрать пункт меню Tools/Java/Generate Java. Ниже приведен код на языке Java, соответствующий классу Customer Information:

```
     //## //## Source file: Customer__Information.java
//## //## Subsystem: Component View
//## //## Module: Customer Information
     //##begin module.cm preserve=no
/*  %X %  %Q %  %Z %  %W % */
//##end module.cm
     //##begin module.cp preserve=no
//##end module.cp
     //##begin module.additionalImports preserve=no
//##end module.additionalImports
     //##begin module.imports preserve=yes
//##end module.imports
     //##begin module.declarations preserve=no
//##end module.declarations
     //##begin module.additionalDeclarations preserve=yes
//##end module.additionalDeclarations
     public class Customer__Information {
//##begin Customer__Information.initialDeclarations
preserve=yes
//##end Customer__Information.initialDeclarations
     public int m_CustomerID;
private int m_Name;
```

```
private int m_Account;
public Vector m_negotates = new Vector();
     public void Check_Account() {
//##begin Customer__Information::Check Account
%3561A0AF032A.body preserve=yes
//##end Customer__Information::Check Account
%3561A0AF032A.body
     }
     //##begin Customer__Information.additionalDeclarations
preserve=yes
//##end Customer__Information.additionalDeclarations
     }
```

При генерации кода Rational Rose включает строки комментария, начинающиеся последовательностью символов «//##». Сгенерированный код (в отличие от кода, сгенерированного ERWin) не является готовым приложением. Здесь генерируются лишь заголовки методов (Check_Account), сами методы необходимо дописывать вручную.

1.7. ARIS-средства описания бизнес-процессов

По данным Gartner Group, лидерами мирового рынка средств описания бизнес-процессов являются системы *ARIS* и *Corporate Modeler Suite* [1], представляемые на мировом рынке компаниями IDS Scheer AG и Casewise. Но, несмотря на лидирующее положение Corporate Modeler Suite на Западе, этот инструмент, в отличие от ARIS, пока еще не столь популярен на российском рынке.

Анализ результатов проектов по внедрению сложных ERP-систем в России и Европе показывает, что наиболее успешные из них имели своей целью не само по себе внедрение информационной системы, а комплексное повышение эффективности деятельности компании через совершенствование ее бизнес-процессов. Проекты внедрения АСУ, реализующие цели автоматизации существующих процессов, не изменяющие бизнеса и не ориентированные на их использование высшим и средним корпоративным управленческим звеном, являются экономически малоэффективными.

По причине высокой сложности процессов, подлежащих автоматизации, необходимо тщательно подходить к задачам постановки требований к ИС и выработки методологии внедрения. Сама по себе автоматизация не решает всех накопившихся проблем и должна следовать за комплексным обследованием архитектуры организации с целью повышения их эффективности и конкурентоспособности. В связи с этим наиболее эффективным путем внедрения сложных информационных систем является следующий:

определение стратегии развития компании и бизнес-целей по основным направлениям;

создание стратегических карт развития компании, содержащих сбалансированные показатели;

создание комплексной модели деятельности компании, отражающей ее ключевые бизнес-процессы и их ресурсное окружение;

оценка эффективности процессов на основании установленных бизнес-целей развития компании (из Balanced Scorecard) и определение их «узких мест»;

проектирование архитектуры ИС на основе созданной модели бизнес-процессов с учетом заданных параметров эффективности всей компании и отдельных ее процессов;

формирование укрупненного плана внедрения ИС;

детальная постановка требований к ИС на основе построенных моделей бизнес-процессов.

Для эффективной реализации предлагаемой идеологии внедрения необходимо использование специализированных технологий, позволяющих не только создать графическое описание модели процессов, но и обеспечить наглядное представление всех взаимосвязей между этими процессами, их связь с бизнес-целями и показателями эффективности, провести формальный анализ процессов по ряду критериев, а также сформировать любой набор документов на основе созданной модели процессов – от должностных инструкций и технологических карт до прототипа технического задания на внедряемую ИС.

Все вышеперечисленные задачи могут быть осуществлены средствами *ARIS* и *Corporate Modeler* (главный в семействе продуктов Corporate Modeler Suite), поставляемыми на российский рынок компаниями «Логика бизнеса» и «ФОРС – Центр разработки». Таким образом, используя эти средства для описания бизнес-процессов, можно, во-первых, наглядно сформулировать требования бизнеса к внедряемым информационным технологиям, используя не только модели процессов, но и рассматривая все аспекты их эффективности (включая описание стратегии развития компании). Во-вторых, путем описания бизнес-процессов обеспечивается возможность автоматического создания основного наполнения технического задания на внедряемую систему (не говоря уже о различных регламентах и возможности процессно-ориентированного обучения персонала работе с ИС). Рассмотрим основные возможности, предоставляемые этими инструментами.

Создание полной и многоаспектной модели деятельности компании

Инструменты ARIS и Corporate Modeler позволяют проводить комплексное описание всех сторон деятельности компании. В ARIS существует пять основных направлений описания: описание процессов, описание функций, описание данных, описание входных/выходных потоков и описание организации. Casewise видит организацию с шести

сторон: структура бизнес-процессов, структура организации, местоположение, структура данных, технологии, приложения.

Независимо от того, какой инструмент используется, описание бизнес-процессов рекомендуется проводить «сверху-вниз». В ARIS самый верхний уровень описания дает диаграмма VAD (Value-added Diagram), которая показывает глобальные бизнес-процессы компании. Затем можно спуститься на второй уровень VAD, на котором эти глобальные процессы детализированы. Третий уровень описания процессов – это матрица выбора сценариев (Process Selection Diagram), где рассматриваются различные варианты протекания процесса. Следующий уровень – пошаговое описание процесса EPC (Event driven Process Chain) с упоминанием всех функций, их исполнителей, необходимых материалов, информационных систем и т. д. Количество уровней детализации не ограничено, но не рекомендуется использовать более четырех-пяти.

Corporate Modeler содержит похожие взгляды на описание архитектуры организации. Рассмотрим 5 основных уровней абстракции, выделяющихся при моделировании организации с применением методологии Casewise Framework, которую можно использовать при работе с Corporate Modeler.

Верхний уровень – это уровень бизнеса в целом. Здесь описываются: стратегические бизнес-цели и критические факторы успеха; бизнес-процессы; основные отделы организации; местоположение бизнес-единиц; важнейшая информация; значительные для бизнеса события. Ниже следует уровень организации, на котором описываются объекты, детализирующие предыдущий уровень. Здесь не уделяется значительное внимание системным или физическим ограничениям. На третьем уровне – уровне систем – описывается работа организации с учетом ограничений, накладываемых бизнесом и ИТ. Этот уровень показывает, как в дальнейшем модели уровня организации будут воплощены в действующие системы. На четвертом уровне – уровне технологий – детально описываются технологии, применяемые в организации. На пятом уровне – уровне деталей – происходит описания реальных объектов, таких как сети, люди, приложения, базы данных.

Corporate Modeler включает основные шесть типов диаграмм:

Hierarchy Diagrams – показывают связи между организационными элементами, процессами, информационными системами, расположением подразделений и данными;

Matrix Diagrammer – служит для управления взаимосвязями между любыми типами и категориями объектов;

Data Flow Diagrammer – диаграммы потоков данных, отражающие движения данных в организации;

Process Dynamics Diagram – описание потока деятельности, содержащее функции, исполнителей, материальные ресурсы, информационных систем, местоположения, документы и т. д.;

Entity Relationship Diagrammer – диаграммы отношений между сущностями, используемые для моделирования данных;

Generic Diagrammer – обеспечивают построение диаграмм в соответствии с любой нотацией.

На каждом уровне абстракции могут применяться любые типы диаграмм, и необходимость их использования трактует Casewise Framework.

Основным отличием Corporate Modeler и ARIS является то, что набор правил, которых необходимо придерживаться при создании конкретной диаграммы, в первом случае не так строг, как во втором. Иначе говоря, Corporate Modeler дает возможность построения собственной методологии. По словам Вадима Крутова, эксперта отдела качества компании «ФОРС – Центр разработки», используя Corporate Modeler, можно выбрать ту методологию, которую удобнее использовать в конкретном случае, в то время как применять ARIS далеко не всегда целесообразно, поскольку она требует серьезных затрат на обучение. Он также отмечает, что научиться работать с ARIS непросто из-за большого количества диаграмм и объектов, и то, что с клиентом, не знакомым с конкретной методологией, необходимо общаться на понятном для него языке. Corporate Modeler, по его мнению, обладает всей мощью графического представления, позволяющей сделать модель интуитивно понятной каждому человеку, при этом сохранив серьезные возможности для анализа.

С другой стороны, в ARIS существуют методологические фильтры, стандартные или создаваемые пользователем, которые разрешают или запрещают использование того или иного типа модели, объекта, связи, атрибута. Таким образом, методологические фильтры позволяют облегчить работу пользователя путем ограничения количества применяемых типов объектов, моделей и т. д. Выбор фильтра зависит от требований пользователя. Например, если необходимо провести динамическое моделирование процесса, то, используя фильтр Simulation/ABC, пользователь получит все модели, связи и объекты, необходимые для проведения динамического моделирования и расчета стоимости процесса. Методологические фильтры особенно удобны для начинающих пользователей, так как они позволяют сосредоточиться именно на построении необходимых моделей.

Еще одно отличие заключается в том, что Corporate Modeler дает возможность создавать свои категории объектов, в то время как в ARIS наборы объектов строго определены. Вадим Крутов утверждает, что при построении организационной структуры предприятия посредством

Corporate Modeler объект типа «organization» может делиться на несколько категорий, в соответствии с которыми отображается структура организации, т. е. объекты делятся на такие уровни, которые удобно использовать. Это может дать более точное описание.

Моделирование бизнес-процессов. Оптимизация

Основная ценность рассматриваемых инструментов заключается не только в возможности создания модели организации, но и в способности к проведению достаточно глубокого анализа бизнес-процессов. После того как создана модель организации «как есть», средства ARIS и Corporate Modeler позволяют взглянуть на организацию и понять, где возникли «узкие места» процессов, какие ресурсы неэффективно используются и какие цепочки бизнес-процессов надо изменить. Модули ARIS Simulation и Process Dynamics Modeler&Simulator, входящие в ARIS и Corporate Modeler, позволяют в динамике рассматривать различные сценарии выполнения бизнес-процессов, не меняя реальной структуры организации. Иначе говоря, с помощью этих модулей можно проводить анализ «а что, если…», тем самым получая возможность находить наиболее оптимальный сценарий прохождения процесса.

Статистика, получаемая по итогам динамического моделирования, дает исчерпывающую информацию о ходе процесса и помогает принимать решения по оптимизации. Таким образом, при моделировании подбираются оптимальные параметры (время исполнения, количество задействованных исполнителей, стоимость процесса) различных бизнес-процессов. В целом функциональность модулей Process Dynamics Modeler&Simulator и ARIS Simulation совпадает.

В ARIS имеется еще один модуль для оценки моделей. ARIS Analysis позволяет проводить классификацию функций в зависимости от значений их атрибутов, выявлять необходимые организационные изменения и анализировать движение информации внутри процесса, использование элементов автоматизированной системы в процессах, затраты времени и средств на их выполнение.

Интеграция с другими программными приложениями

Немаловажным аспектом применения средств описания бизнес-процессов является возможность их интеграции с другими программными приложениями. Интеграция необходима для более полного использования возможностей систем моделирования, когда информация из моделей бизнес-процессов может быть передана в другие системы, например, в workflow или системы контроля за бизнесом.

Часто используется интеграция с Case-средствами для создания кода баз данных или разработки собственных информационных систем.

ARIS имеет множество интерфейсов с такими продуктами, как COOL:Biz, COOL:BusinessTeam, COOL:Enterprise Advantage, Gen AllFusion, ERwin, Oracle Designer, PowerDesigner/PowerAMC, Rational

Rose, Select Enterprise, Select SE (Systems Engineer), System Architect case/4/0. Также существуют интерфейсы с workflow-системами Ultimus, Staffware, приложениями MS Office, Lotus-Notes.

Casewise имеет интерфейсы с инструментами управления требованиями Telelogic DOORS, Rational RequsitePro, объектно-ориентированным инструментом проектирования Rational Rose, инструментами моделирования баз данных Oracle Designer, ERwin, Sybase PowerDesigner. Полноценно реализованы XML Export/Import, а также интеграция с workflow-системой TIBCO InConcert, приложениями MS Office и MS Visio.

Индивидуальные особенности

Каждый из рассмотренных инструментов предоставляет дополнительные возможности.

Так, в ARIS имеется широкий набор scout'ов, содержащих описание процедуры, методологию и инструментальные средства для проведения проектов определенного типа:

создание системы менеджмента качества (Quality Management Scout);

создание системы управления операционными рисками (Process Risk Scout);

внедрение сложных информационных систем (Software Engineering Scout);

ре-документирование внедренной системы mySAP (Re-Documentation Scout).

Кроме того, ARIS содержит аналитический инструмент Process Performance Manager (PPM), который интегрируется в функционирующие в компании информационные системы и позволяет всесторонне анализировать протекающие там процессы и их ключевые показатели, делать аналитические выборки и искать факторы, в наибольшей степени влияющие на качество осуществления процессов. Таким образом, при помощи ARIS PPM появляется возможность настроить мониторинг процессов, проходящих через различные информационные системы, что позволяет оценить качество внедрения систем и соответствие их реального функционирования спроектированным ранее процессам.

1.8. Средства моделирования бизнес-процессов, приложений и данных

Моделирование и проектирование данных и приложений основывается на сформулированных требованиях и является весьма важной частью процесса создания готового продукта. Инструменты для поддержки этого этапа жизненного цикла приложений можно условно разделить на средства моделирования бизнес-процессов, средства проектирования данных и средства объектно-ориентированного моделирования. Отметим, что сегодня многие компании производят все

три категории инструментов, интегрирующихся между собой (например, позволяющих сгенерировать модель данных и модели бизнес-процессов или синхронизировать их между собой), либо реализуют функциональность нескольких разнотипных средств моделирования в одном продукте, поэтому применительно к указанной категории инструментов имеет смысл говорить о линейках продуктов различных производителей.

Ниже перечислены наиболее известные на российском и мировом рынке продукты и линейки продуктов, предназначенные для *моделирования и проектирования [59]*.

CA ERwin Modeling Suite 7.3 – мощная линейка интегрированных CASE-средств, которые позволяют моделировать различные аспекты деятельности предприятия и проектировать информационные системы. В нее входят:

CA ERwin Process Modeler (BPwin) 7.3 – средство функционального моделирования бизнес-процессов;

CA ERwin Data Modeler (ERwin) 7.3 – проектирование, документирование и сопровождение баз данных и хранилищ данных;

CA ERwin Data Profiler 7.3 – решение для анализа и профилирования данных, обеспечивает эффективную обработку исходной информации;

CA ERwin Data Model Validator (ERwin Examiner) 7.3 – проверка структуры баз данных и качества моделей AllFusion ERwin Data Modeler;

CA ERwin Model Manager (ModelMart) 7.3 – среда для совместного моделирования в CA ERwin Data Modeler и/или CA ERwin Process Modeler.

Дополнительными элементами являются *CA ERwin Saphir Option* и *CA ERwin Model Navigator*.

В России эта линейка продуктов весьма популярна, а AllFusion ERwin Data Modeler является одним из самых распространенных средств проектирования данных благодаря поддержке широкого спектра СУБД.

Oracle Designer представляет собой инструмент, позволяющий проектировать данные, моделировать бизнес-процессы, создавать диаграммы потоков данных и функциональные модели, а также реализовывать их в виде серверных объектов. Этот продукт предназначен главным образом для применения совместно с СУБД Oracle и поддерживает все ее особенности, хотя с его помощью можно осуществлять и обратное проектирование для СУБД других производителей.

Sybase PowerDesigner – это инструмент, в состав которого входят средства создания моделей и объектно-ориентированного моделирования. Помимо серверных СУБД производства Sybase, PowerDesigner способен работать с любыми ODBC-источниками, генерировать код клиентских приложений для PowerBuilder, а также классы Java и компоненты JavaBeans. Возможно и обратное проектирование диаграмм классов из

исходных текстов Java. Набор функциональных возможностей продукта варьируется в зависимости от редакции. Отметим, что PowerDesigner весьма популярен на российском рынке, и отнюдь не только среди пользователей СУБД и средств разработки Sybase.

System Architect является универсальным продуктом, позволяющим осуществлять не только проектирование данных, но и структурное моделирование. В его состав входят средства проектирования данных и создания ER-диаграмм, а также обеспечивается поддержка СУБД практически всех ведущих производителей. Компоненты System Architect позволяют документировать процесс работы над проектом, включая техническое задание, план тестирования и др. С помощью System Architect возможно генерировать код клиентских приложений для Visual Basic, Delphi и PowerBuilder, классы C++.

Microsoft Office Visio 2007 представляет собой универсальное средство моделирования данных и приложений, поддерживающее и создание моделей данных, и объектно-ориентированное моделирование приложений. Как и подавляющее большинство подобных продуктов, Visio позволяет производить прямое и обратное проектирование данных, поддерживает все ODBC- и OLE DB-источники данных и особенности серверных СУБД всех ведущих производителей. Помимо средств проектирования данных, Visio включает средства объектно-ориентированного моделирования и генерации кода приложений (главным образом для платформы Microsoft .NET).

IBM Rational Rose – одно из самых популярных средств объектно-ориентированного UML-моделирования приложений. Этот продукт позволяет решать практически любые задачи в проектировании информационных систем – от анализа бизнес-процессов и моделирования данных до генерации кода на различных языках программирования, а также обладает средствами интеграции с другими инструментами Rational, в частности с Requisite Pro.

Rational XDE Professional (IBM) – инструмент UML-моделирования, встраиваемый в среды разработки Microsoft Visual Studio.NET и IBM WebSphere Studio Application Developer. Этот продукт дает возможность осуществлять визуальное проектирование на основе диаграмм UML и по окончании процесса проектирования генерировать код на выбранном языке программирования, а также проводить двустороннюю синхронизацию кода и модели.

Borland Together является платформой для анализа и проектирования приложений, интегрирующейся с различными средствами разработки как самой компании Borland, так и других производителей (в частности, Microsoft). Продукт позволяет осуществлять моделирование и проектирование приложений и данных, причем степень его интеграции со средствами разработки в настоящее время такова, что изменения модели

данных приводят к автоматическому изменению кода приложения, равно как и изменения в коде приводят к изменению в моделях (указанная технология интеграции инструментов моделирования и средств разработки получила название LiveSource).

Из вышесказанного следует, что основной тенденцией развития средств моделирования в данный момент является активное предложение средств интеграции их между собой и с инструментами поддержки других этапов жизненного цикла приложений: средств разработки, средств управления требованиями и средств управления изменениями. Происходившее в последние годы слияние компаний, специализирующихся на производстве подобных инструментов, позволило создать линейки продуктов, в своей совокупности реализующих все или почти все задачи, которые могут возникнуть на этапе проектирования приложений.

Еще одной тенденцией можно назвать появление таких средств моделирования, которые тесно интегрированы со средствами разработки не только на уровне синхронизации кода и модели, но и на уровне полного определения поведения приложения непосредственно в самой модели (как это, например, реализовано в архитектуре Model Driven Architecture).

Ключевые характеристики CA ERwin Modeling Suite 7.3 (ERwin Modeling Suite):

двунаправленная синхронизация моделей/баз данных;

генерирование проектных решений;

прямое и обратное проектирование баз данных и хранилищ;

моделирование бизнес-процессов и синхронизация с моделями данных;

средства управления, публикации и оптимизации моделей.

Отличительные возможности и функции CA ERwin Modeling Suite 7.3

Моделирование данных и проектирование БД. CA ERwin Data Modeler (CA ERwin DM) – это ведущее решение для моделирования данных, которое позволяет создавать и поддерживать БД, хранилища данных и модели ресурсов корпоративных данных. Модели данных помогают визуализировать структуры данных, благодаря чему возможно организовывать данные и управлять ими, а также справляться со сложностями, связанными с данными, технологиями БД и средой развертывания.

Полное сравнение. Управление архитектурными изменениями модели и базы данных с помощью двунаправленной синхронизации объектов и связанных с ними физических свойств, а также их идентификация и оценка. Не нужно ни кодирование, ни знание структуры баз данных. При изменении модели в результате синхронизации

автоматически генерируется сценарий изменений, которые вносятся в базу данных.

Генерация дизайна базы данных. Широкофункциональный макроязык, не зависящий от базы данных, и шаблоны целостности ссылочных данных автоматизируют генерацию объектов баз данных и облегчают настройку сложных хранимых процедур, сценариев и триггеров.

Генерация модели данных на основе анализа существующей БД. Недокументированную информацию, которая содержится в сценариях на языке SQL или в базе данных, можно визуализировать или повторно использовать для создания новых моделей данных и/или объектов базы данных.

Многократно используемые стандарты/объекты дизайна. CA ERwin Data Modeler предоставляет широкий набор многократно используемых объектов моделирования. Это позволяет создавать, поддерживать и применять схемы преобразования имен, схемы соответствий типов данных, шаблоны для генерации схем, доменные определения и базу для других стандартов моделирования в масштабе организации.

Отчеты и публикация метаданных. CA ERwin Data Modeler предоставляет гибкие, настраиваемые и многократно используемые возможности для создания отчетов и вывода на печать. В этих отчетах для эффективного обмена данными и совместной работы при использовании данных и БД и управлении ими интегрируются графические модели и метаданные.

Многократное использование расширенной модели данных. CA ERwin Data Modeler обеспечивает обмен моделями данных и связанными с ними метаданными в замкнутом цикле, а также их преобразование. При этом применяются различные стандарты метаданных (XSD, XMI, CWM), закрытые хранилища метаданных в областях бизнес-аналитики, перемещение/интеграция данных (ETL/EAI/EII) и управление метаданными, а также современные продукты для моделирования, охватывающие данные, язык UML и корпоративную архитектуру.

Совместное моделирование. CA ERwin Model Manager (CA ERwin MM) представляет собой масштабируемую многопользовательскую среду моделирования, которая обеспечивает эффективную совместную работу специалистов по моделированию. Этот пакет служит средством интеграции для двух ведущих средств моделирования от компании CA – CA ERwin DM и CA ERwin Process Modeler. CA ERwin Model Manager облегчает многопользовательский обмен данными в среде рабочей группы, что способствует повышению уровня организации совместной работы, качества и производительности.

Параллельный доступ к моделям. Благодаря опциональной блокировке моделей CA ERwin Model Manager позволяет нескольким пользователям работать с одной определенной моделью. При этом управление интеграцией и целостностью объединенных результатов такой деятельности осуществляется по требованию.

Возможности разрешения коллизий при многопользовательской работе. Для обеспечения непрерывной целостности модели конфликтующие изменения в ней определяются и управляются автоматически. Пользователи продукта CA ERwin Model Manager автоматически получают уведомление о существовании и природе конфликтов при моделировании. Для максимизации производительности труда коллектива и обеспечения целостности модели пользователей уведомления проводят пользователей через процесс разрешения конфликтов.

Анализ последствий изменений в модели и управление версиями. CA ERwin Model Manager предоставляет собой подробную сводку изменений, показывающую последствия для всех затронутых объектов модели. Это помогает выбрать, какие изменения применить к репозиторию модели. CA ERwin Model Manager позволяет хранить и анализировать историю модели и всех изменений, осуществляемых в ней, а также обеспечивает доступ к ней. На основе этой информации можно при необходимости восстановить модели до предыдущего состояния.

Гибкое управление доступом к репозиторию. CA ERwin Model Manager эффективно использует авторизацию как средствами операционной системы, так и средствами системы управления базами данных (СУБД). Вне зависимости от выбора пользователя администратору предоставляется простые, но мощные возможности управления доступом. Администрирование на базе профилей управляет доступом пользователей к моделям и объектам с различным уровнем гранулярности.

Интеграция процессов и данных. CA ERwin Process Modeler (CA ERwin PM) представляет собой мощное средство моделирования, которое в едином пакете поддерживает моделирование процессов, потоков данных и потоков работ. Этот продукт служит нуждам как бизнес-аналитиков, так аналитиков технологий. Пакет многократно использует информацию о моделировании с этих трех точек зрения для обнаружения точек конфликтов и пробелов. В конечном счете это помогает выяснить окружение для разработки БД и приложения. CA ERwin Process Modeler предоставляет механизм сбора ключевых знаний об организации и бизнесе, что повышает производительность труда и качество, а также позволяет руководить процессом разработки приложения.

Поддержка различных методик моделирования процессов. CA ERwin Process Modeler предоставляет интегрированные средства для многократного использования и координации для методик моделирования

бизнес-процессов (IDEF0), технологических процессов (IDEF3) и потоков данных (DFD).

Диаграммы типа «плавательная дорожка». Предоставляют эффективный механизм для визуализации и оптимизации сложных процессов и организуют процессы параллельно функциональным границам, что позволяет одновременно просматривать внутренние зависимости и организационные взаимодействия.

Интеграция с CA ERwin Data Modeler. Интеграция между CA ERwin Process Modeler и CA ERwin Data Modeler помогает разрешать проблемы, связанные с анализом корпоративной архитектуры. Встроенные функции использования и поддержки метаданных в CA ERwin Process Modeler обеспечивают подробное соотнесение данных с процессом по принципу «где используется».

Интеграция с CA ERwin Model Manager. CA ERwin MM представляет собой многопользовательский репозиторий моделей. Он обеспечивает централизованное хранение моделей, контроль доступа, управление версиями и службы создания отчетов для CA ERwin Process Modeler, а также для CA ERwin Data Modeler.

Проверка и оптимизация дизайна БД. CA ERwin Data Model Validator (CA ERwin DMV) автоматизирует задачу проверки схемы базы данных на предмет соответствия четко определенным реляционным правилам, так как проверка, выполняемая вручную, является дорогостоящей и трудоемкой. Каждое изменение в схеме может неблагоприятно повлиять на дизайн базы данных, привести к ее повреждению и падению производительности. Автоматизация этой задачи позволяет проектировщикам базы данных эффективно реализовать преимущества проверки дизайна, одновременно в сжатые сроки минимизируя негативные последствия.

Всеобъемлющая диагностика и отчетность. CA ERwin Data Model Validator анализирует структуру данных в схеме, ключи, индексы, поля и связи на предмет нарушения реляционной теории. Это средство генерирует графическую документацию всей структуры БД, включая перекрестные ссылки между столбцами и списки отношений. Диагностику можно настраивать, выбирая важнейшие для организации и проверки результаты, которые упорядочены по категории или степени серьезности.

Выделение противоречий в дизайне. CA ERwin Data Model Validator предоставляет подробные отчеты, которые могут повысить производительность труда благодаря ускорению процесса проверки дизайна. Новаторская функция «Покажи мне» выделяет определенные проблемы дизайна в сложных моделях БД. Это исключает ручной поиск проблем в большой модели. Проверку также можно применить к базе данных или к подмножеству моделей.

Функция «Научи меня». CA ERwin Data Model Validator поясняет теорию, лежащую в основе нарушений реляционных правил. Функция «Научи меня» выявляет последствия выбора дизайна или его модификаций в свете реляционной теории. Инструктируя специалистов по моделированию данных насчет последствий их решений, CA ERwin DMV помогает им создавать базу данных более высокого качества. Также этот продукт является прекрасным средством для обучения нового персонала теории реляционных баз данных.

Интеграция метаданных для пакетированных приложений. CA ERwin Saphir Option извлекает и сохраняет подробные метаданные из таких сложных приложений, как SAP R/3, SAP BW, Oracle PeopleSoft Enterprise, Oracle JD Edwards Enterprise One и Oracle Siebel. Перед созданием моделей CA ERwin Data Modeler специалисты по моделированию могут искать и просматривать метаданные, а также делить их на подмножества. Эти специфичные для систем ERP модели в CA ERwin Data Modeler можно затем использовать для поддержки различных инициатив, например, создания специализированных отчетов, организации хранилищ данных, управления метаданными, а также управления пакетами и их настройки.

Доступ к метаданным определенного пакета. CA ERwin Saphir Option предоставляет экспертное представление внутренней работы пакетированного приложения и информации, с которой оно работает. Метаданные, которые собираются и публикуются с помощью CA ERwin Saphir Option, также содержат любые настройки, присущие определенному экземпляру системы планирования ресурсов предприятия (ERP). Такие артефакты дизайна определенного производителя, как домены и описатели полей, обеспечивают уникальное представление основных элементов сложных бизнес-систем.

Удобный браузер данных. CA ERwin Saphir Option обеспечивает доступ к таблицам, полям, связям, элементам данных, представлениям, доменам и индексам. При этом требуются лишь незначительные знания о конкретном приложении. Пользователи могут просматривать метаданные, организованные в соответствии с иерархией пакетированного приложения либо выбирть отдельные таблицы и/или представления и следовать заданным внутренним связям.

Мощные возможности поиска и создания запросов. Позволяют быстро находить таблицы и поля, а также использовать преимущества функций расширенного поиска. Пользователи могут находить все таблицы, которые содержат определенное поле, либо все таблицы с определенной строкой текста в их названии. Функция QBE в CA ERwin Saphir Option также сопровождает пользователей при навигации по пакетированному приложению.

Экспорт из CA ERwin Data Modeler. Продукт CA ERwin Saphir Option полностью интегрирован с CA ERwin Data Modeler. С помощью CA ERwin Saphir Option пользователи могут легко создавать модели данных в формате CA ERwin Data Modeler. Это позволяет эффективно использовать возможности визуализации, анализа, проектирования и генерации ведущего решения для моделирования данных от компании CA.

Ключевые характеристики CA ERwin Data Modeler 7.3:

синхронизация моделей/баз данных;

автоматизированное создание структуры базы данных и обратное проектирование;

публикация моделей;

поддержка нотаций IDEF1x, IE, Dimensional;

возможность совместной работы группы проектировщиков (с помощью среды CA ERwin Model Manager (ModelMart));

документирование структур баз данных;

возможность переноса структур баз данных (но не самих данных) из одного типа СУБД в другой.

CA ERwin Data Modeler 7.3 (ERwin) необходим всем компаниям, разрабатывающим и использующим базы данных, администраторам баз данных, системным аналитикам, проектировщикам БД, разработчикам, руководителям проектов.

Дополнительные аргументы для разработчиков ПО

Позволяет получить точную и наглядную информацию, где хранятся данные и как получить к ним доступ, описать структуру БД, используя визуальные средства, а затем автоматически сгенерировать файлы данных для любого типа СУБД.

Дополнительные аргументы для администраторов баз данных

Позволяет максимально повысить производительность информационной системы благодаря поддержке работы с БД на физическом уровне, учитывая особенности каждой конкретной СУБД, и уменьшает число одинаковых операций, облегчая и сокращая время работы.

Дополнительные аргументы для руководителей

Повышает гибкость и эффективность организации за счет возможности быстрой адаптации базы данных к меняющимся потребностям рынка, а использование профессиональных средств является фактором конкурентной борьбы.

Дополнительные аргументы для руководителей проектов

ERwin помогает тщательно задокументировать структуру БД, позволяет получить отчеты презентационного качества, а также с помощью CA ERwin Model Manager возможно эффективное управление ходом проектирования.

Дополнительные аргументы для крупных компаний

Возможность документирования собственной ИС для ее идентификации и бесприпятственного перехода на другие СУБД.

Дополнительные аргументы для системных интеграторов

Поддержка различных типов СУБД (CA ERwin Data Modeler для СУБД более 20 производителей) и возможность адаптации к изменяющимся требованиям заказчика.

Новые функции CA ERwin Data Modeler 7.3 (ERwin)

Улучшенная поддержка баз данных. Организации должны использовать свои инвестиции в область информационных технологий для успешной поддержки информационных требований бизнеса. Тесная интеграция и поддержка «добавочной ценности» нескольких СУБД является основой оптимизации анализа и проектирования в организациях при помощи CA ERwin Data Modeler. В соответствии с приоритетами по поддержке заказчиков в CA ERwin Data Modeler теперь имеется улучшенная полная поддержка SQL Server 2005 и новых баз данных MySQL, DB2 UDB v9 и Sybase IQ. Поддержка баз данных включает прямое проектирование (FE), обратное проектирование (RE), полное сравнение (CC).

*Интеграция с VSDB PRO***.** Интеграция CA ERwin Data Modeler с Microsoft»s Visual Studio Team Edition for Database Professionals (VSDB Pro) объединяет ведущие разнородные системы проектирования баз данных. Основное внимание уделено разработке для SQL Server 2005 и среде управления жизненным циклом приложений. Такая интеграция предоставляет полную платформу, позволяющую организациям управлять разработкой и развертыванием важных приложений SQL Server 2005. При этом возможны:

визуальное проектирование, определение и создание управляемых объектов VSDB Pro разработчиками моделей;

обратное проектирование существующих моделей CA ERwin Data Modeler в управляемые объекты VSDB Pro, включая извлечение информации о схемах из баз данных, отличных от SQL Server 2005;

прямое проектирование управляемых объектов VSDB Pro в CA ERwin Data Modeler в целях визуализации и создания отчетов по метаданным.

Улучшенные функции полного сравнения в CA ERwin Data Modeler 7.3. Представлено несколько улучшений, позволяющих упростить однопроходную функцию полного сравнения, что делает ее использование более удобным.

Глобальный поиск и замена. Метаданные модели, с которыми довольно сложно начинать работать, значительно влияют на функции, применяемые пользователями при проектировании модели данных (например, вычисляемые значения, наследование, макросы и замещения). Несмотря на всю сложность и исключительность моделирования,

пользователи по-прежнему нуждаются в функции эффективного и простого управления изменениями в больших корпоративных моделях. Такая функция отличается возможностью глобального поиска определенной информации в метаданных моделей для оценки влияния предложенных изменений и глобального обновления объектов моделей для обеспечения согласованного управления изменениями метаданных. В CA ERwin Data Modeler 7.3 входит новая контекстная функция глобального поиска и замены.

Фильтрация журнала работ. В журнале работ регистрируются все изменения модели, сделанные в сеансе моделирования. Это важная составляющая часть функционала CA ERwin Data Modeler по расширению возможностей проведения работ по моделированию. Кроме того, журнал работ представляет собой ключевую функцию, позволяющую пользователям оценить влияние изменения (или планируемого изменения) и составить документацию по изменениям моделей с точки зрения согласованности. Благодаря функции «фильтрации» журнала работ в CA ERwin Data Modeler 7.3 пользователи могут динамически ограничивать объем отображаемых метаданных и, следовательно, повышается удобство использования информации журналов.

Изменение пользовательского интерфейса дискриминантов подтипа. Для полного соответствия и поддержки спецификаций моделирования IDEF1x CA ERwin Data Modeler позволяет пользователям выбирать атрибуты из других объектов супертипов и использовать их в качестве дискриминаторов отношений подтипов.

Функциональные возможности CA ERwin Data Modeler 7.3 (ERwin)

Многоуровневая архитектура проектирования. CA ERwin Data Modeler позволяет гибко генерировать модели данных, удовлетворяющие заданным требованиям. Поддерживается разделение логических и физических моделей, а также традиционные объединенные логические/физические модели. Обеспечивается хранение истории решений по дизайну и знаний об отношениях, что позволяет быстро решать проблемы воздействия изменений с одного уровня дизайна на другой (рис. 1.47).

С помощью CA ERwin Data Modeler можно создавать раздельные логические и физические модели данных, а также новые модели, основанные на существующих моделях и структурах.

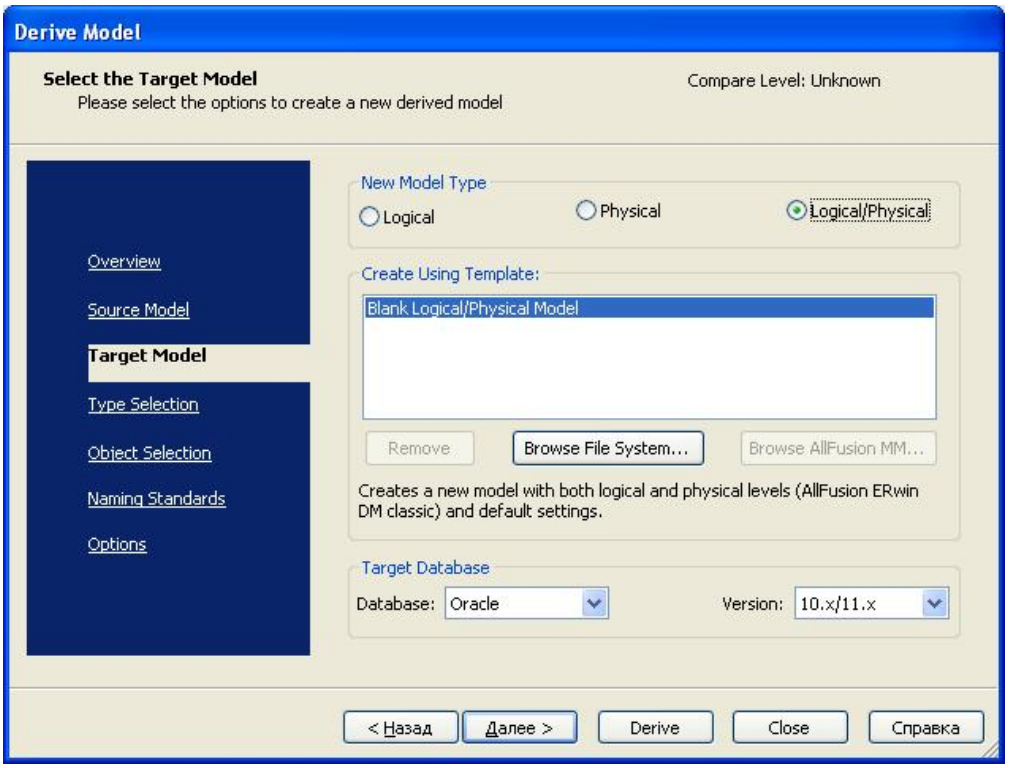


Рис. 1.47. Многоуровневая архитектура проектирования

Технология преобразования. Физический дизайн БД редко совпадает с изначальным логическим дизайном. Ограничения, накладываемые бизнесом, диктуют необходимость денормализации таблиц, связанную с современными требованиями к производительности (SOA). Технология преобразования обеспечивает внедрение этих изменений при сохранении целостности исходного дизайна.

Определение стандартов. Определение и сохранение стандартов поддерживается через доменный словарь (Domain Dictionary), редактор стандартов наименований (Naming Standards Editor) и редактор стандартов типов данных (Datatype Standards Editor). Доменный словарь содержит многократно используемые атрибуты и помогает обеспечить целостность имен и определений во всей модели. Редактор стандартов наименований позволяет создать глоссарий, аббревиатуры и правила наименования, которые также можно многократно использовать в модели. Редактор стандартов типов данных дает возможность определить стандарты для соотнесения типов данных, назначенных как пользователем, так и по умолчанию, с определенными системными типами данных для управления БД.

Управление большими моделями. CA ERwin Data Modeler помогает управлять большими корпоративными моделями с помощью предметных областей (Subject Areas) и хранимых отображений (Stored Displays). Предметные области предоставляют сфокусированное представление для отдельных специалистов по моделированию, делящее модель на меньшие

управляемые подмножества. Хранимые отображения предлагают различные графические представления модели или ее предметных областей, что облегчает обмен информацией между специализированными группами пользователей.

Полное сравнение. Эта мощная функция автоматизирует полную двунаправленную синхронизацию моделей, сценариев и БД. Она сравнивает один элемент с другим, отображает различия и позволяет выбрать, какие из этих различий перемещать и в каком направлении. Если изменения модели переносятся в БД, CA ERwin DM может по желанию разработчика автоматически сгенерировать сценарий ALTER для этой БД.

*Генерация дизайна баз данных***.** CA ERwin Data Modeler включает в себя оптимизированные шаблоны триггеров целостности ссылочных данных и широкофункциональный межплатформенный макроязык для БД, позволяющий настраивать триггеры и хранимые процедуры. Настраиваемые шаблоны облегчают создание физического дизайна и полных определений (в соответствии с конечной БД).

*Проектирование хранилищ и витрин данных***.** Производительность, удобство использования и значение хранилища данных определяется лежащим в его основе дизайном. CA ERwin Data Modeler поддерживает специализированные технологии для хранилищ данных (например, многомерное моделирование Star Schema и Snowflake), оптимизируя таким образом хранилище для заданных нужд производительности и анализа. Также продукт собирает и документирует широкий спектр информации о хранилище, включая источники данных, логику преобразования и правила управления данными.

Отчеты и распечатки. Для обмена данными и совместной работы в процессе моделирования данных очень важны визуализация и публикация (рис. 1.48). CA ERwin DM предоставляет гибкие и настраиваемые возможности для создания отчетов и вывода на печать. Отчеты можно генерировать в различных форматах, включая HTML, PDF, RTF и TXT.

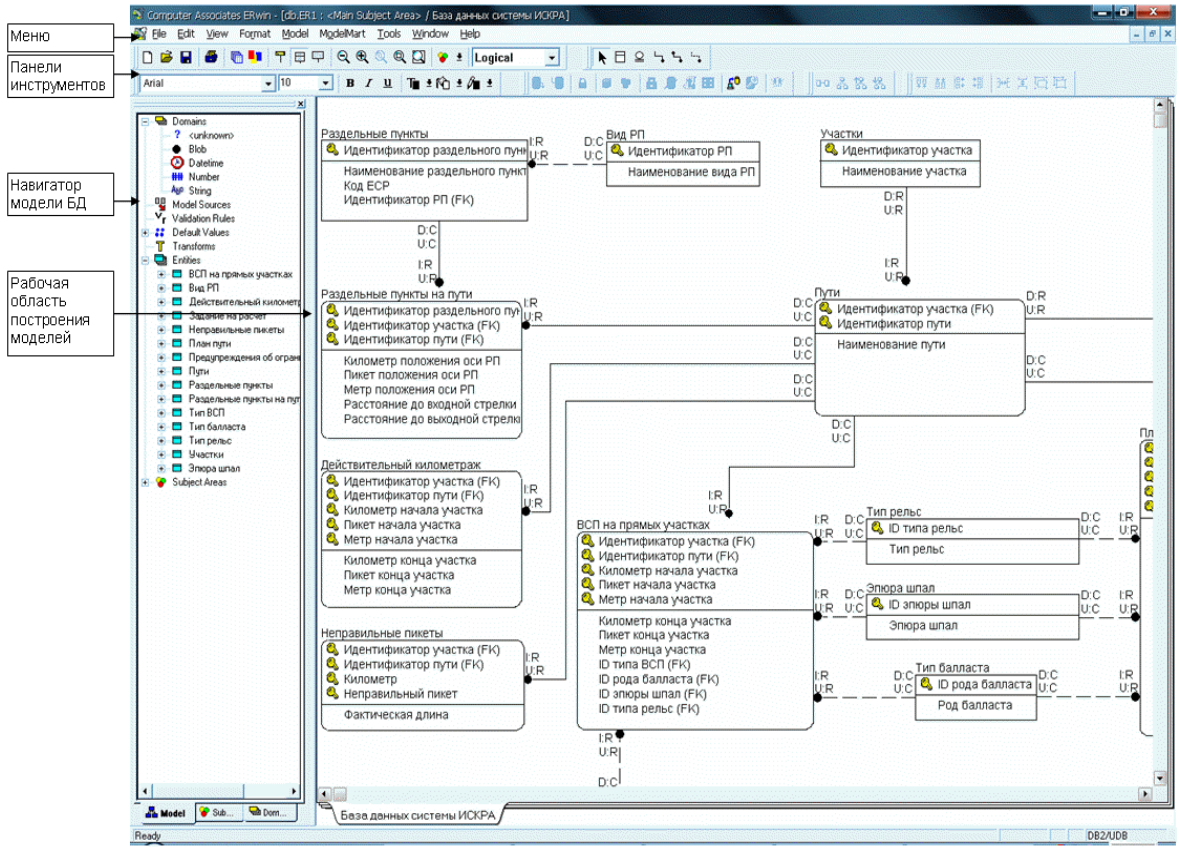


Рис. 1.48. Модель данных и визуализация дизайна БД

Графическое изображение объектов обеспечивает оптимальную визуализацию модели и объектов, улучшает читаемость модели и помогает в привязке к бизнес-правилам и конструкциям.

Полнота, открытость и интеграция. Польза от CA ERwin Data Modeler может значительно возрасти благодаря применению других тесно интегрированных продуктов из семейства CA ERwin Modeling Suite, а также интегрированных на уровне метаданных продуктов сторонних производителей.

Интеграция и обмен метаданными. CA ERwin Data Modeler предоставляет возможности импорта и экспорта с помощью готового набора мастеров для обмена метаданными модели. Поддерживаются более 70 стандартных отраслевых форматов для моделирования, дизайна и управления метаданными, включая UML, OMG, CWM, XML, ETL, EII, BI и различные решения для репозиториев метаданных.

CA ERwin® Saphir Option. Это решение извлекает и сохраняет подробные метаданные из таких сложных приложений, как SAP R/3, SAP BW, Oracle PeopleSoft Enterprise, Oracle JD Edwards Enterprise One и Oracle Siebel. Перед созданием моделей CA ERwin Data Modeler специалисты по моделированию могут искать метаданные, просматривать и делить их на подмножества. Эти специфичные для систем ERP модели в CA ERwin Data Modeler можно затем использовать для поддержки различных инициатив:

создания специализированных отчетов, организации хранилищ данных, управления метаданными, а также для управления и настройки пакетов.

CA ERwin Data Model Validator (*CA ERwin DMV*). Этот полезный дополнительный модуль позволяет специалистам по моделированию автоматизировать проверку дизайна БД, что способствует более быстрому и точному выполнению этой задачи. CA ERwin DMV предоставляет пользователям выбор из 80 упорядоченных процедур диагностики, которые можно применить к моделям данных, а также базу знаний, советы по проектированию, предложения по корректировке и сценарии, которые можно применить к модели и связанной с ней БД.

CA ERwin DM Application Program Interface (*API*) *and Add-In Manager*. Позволяет работать с дополнительными программными приложениями, которые поставляют сторонние производители. Это может быть нужно для выполнения определенных задач, выходящих за границы возможностей, которые предоставляются в рамках CA ERwin Modeling Suite.

Упрощение и расширение возможностей дизайна БД. Парадигма визуального проектирования схемы («моделирование данных») является принятым в отрасли методом обеспечения эффективного создания и сопровождения структур БД. Организации, которые используют моделирование данных, быстрее получают отдачу от своих инвестиций, сокращают риск возникновения ошибок и используют ресурсы на протяжении всего жизненного цикла БД более эффективно. Кроме того, CA ERwin Data Modeler позволяет использовать метаданные объектов совместно с другими продуктами, задействованными в организации. Простой графический интерфейс «укажи и щелкни» и солидный список возможностей – это основные компоненты, которые помогают организации создавать высококачественные БД непрерывно, вовремя и в пределах бюджетных ограничений.

Поддерживаемые СУБД:

Oracle;
DB2/UDB (включая iSeries);
SQL Server;
Teradata;
ODBC;
Sybase;
Informix;
Ingres;
Progress;
Access.

Интеграция с другими продуктами

CA ERwin Data Modeler интегрирован с широким спектром сред моделирования, такими как Rational Data Architect, Oracle Designer, Sybase Power Designer и др. – всего 100 популярных продуктов.

Возможности ERwin Data Modeler дополняет линейка продуктов для поддержки всех стадий разработки ИС – CA ERwin от Computer Associates. Линейка CA ERwin включает *инструменты моделирования, управления процессами, изменениями и конфигурациями.*

Линейка взаимно интегрированных *CASE-средств* CA ERwin Modeling Suite включает CA ERwin Data Modeler, CA ERwin Process Modeler для моделирования бизнес-процессов, CA ERwin Data Model Validator для проверки моделей баз данных. Следует обратить внимание, что CA ERwin Data Model Validator дополняет функциональность ERwin Data Modeler, позволяя искать ошибки в моделях ERwin и в структуре баз данных при одновременном обучении моделированию благодаря режиму подсказок.

CA ERwin Data Model Validator 7.3 (ERwin Examiner)

Проверка структуры баз данных и качества моделей CA ERwin Data Modeler

CA ERwin Data Model Validator **–** инструмент для проверки структуры баз данных и моделей, создаваемых в CA ERwin Data Modeler, позволяющий выявлять недочеты и ошибки проектирования. Его гибкость заключается в том, что можно проводить выборочные тесты, а также анализировать отдельные таблицы. Продукт дополняет функциональность CA ERwin Data Modeler, автоматизирует трудоемкую задачу поиска и исправления ошибок, одновременно повышая квалификацию проектировщиков баз данных, благодаря встроенной системе обучения.

Встроенные функциональные возможности обеспечения качества и проверки моделей позволяют осуществлять контроль семантики моделей на каждой стадии разработки и вносить необходимые исправления, что помогает разработчикам создавать модели высокого качества.

С помощью CA ERwin Data Model Validator можно анализировать структуры данных, ключи, индексы, столбцы и отношения. Кроме того, результаты контроля помогают отобразить в графическом виде структуру всей базы данных, включая столбцы с перекрестными ссылками и списки отношений.

CA ERwin Data Model Validator 7.3 нужен всем компаниям, разрабатывающим и использующим базы данных, администраторам баз данных, системным аналитикам, проектировщикам БД, разработчикам и руководителям проектов.

Преимущества использования CA ERwin Data Model Validator 7.3:

возможность прямой и обратной проверки структур: при помощи продукта можно оптимизировать структуры существующих баз данных или проверять на корректность модели, созданные в CA ERwin Data Modeler;

дополнение функциональности CA ERwin Data Modeler, автоматизация сложных рутинных функций проверки;

встроенная система подсказок предлагает варианты исправления выявленных ошибок и методы повышения эффективности баз данных.

Средства диагностики и проверки CA ERwin Data Model Validator используются для контроля структурной целостности моделей данных CA ERwin Data Modeler или кода SQL/DDL путем применения правил реляционной технологии. CA ERwin Data Model Validator помогает обнаруживать дефекты проектирования, выдает рекомендации корректирующих действий и автоматически генерирует сценарии для реализации выбранных корректировок.

CA ERwin Data Model Validator анализирует структуры данных, ключи, индексы, поля столбцов и отношения на предмет нарушений правил проектирования реляционных баз данных. CA ERwin Data Model Validator выдает детализированные диагностические отчеты, помогающие увеличивать продуктивность за счет ускорения процесса анализа.

Поддерживает платформы Windows, Oracle, MS SQL Server, IBM DB2 UDB, Sybase и ODBC (базы данных).

1.9. Объектно-реляционное моделирование в Power Designer

Объектно-реляционную парадигму поддерживает современное средство Power Designer, в котором реализованы концепции итерационного и структурного проектирования информационной системы.

Итерационный подход предполагает разработку программного обеспечения по спиральной модели. Каждой «виток» спирали включает часть или все этапы проекта (анализ, проектирование, разработка, тестирование, внедрение и сопровождение) и соответствует шагу к «улучшению» программного продукта – наращивание функциональных возможностей, исправление ошибок и т. д. На каждом шаге модификация проекта может проводиться либо на различных уровнях (в программном коде, OOM, CDM, PDM и т. д.), либо в нескольких местах одновременно (при параллельной разработке), при этом для обеспечения целостности всего проекта требуется:

согласовать и протоколировать изменения с помощью репозитория (repository);

обновить связанные модели путем прямого/обратного инжиниринга (forward/reverse engineering);

разрешить конфликты (функции merge/compare/check models).

Сущность структурного подхода заключается в декомпозиции всего приложения на компоненты по функциональным или другим критериям. Каждый компонент, в свою очередь, разбивается на меньшие, и т. д. до выделения бизнес-логики в методах класса. Например, структурирование приложения обеспечивается с помощью иерархического представления объектно-ориентированного моделирования (ООМ) (вложенные диаграммы UML).

Схема возможных преобразований моделей и генерации программного кода представлена на рис. 1.49.

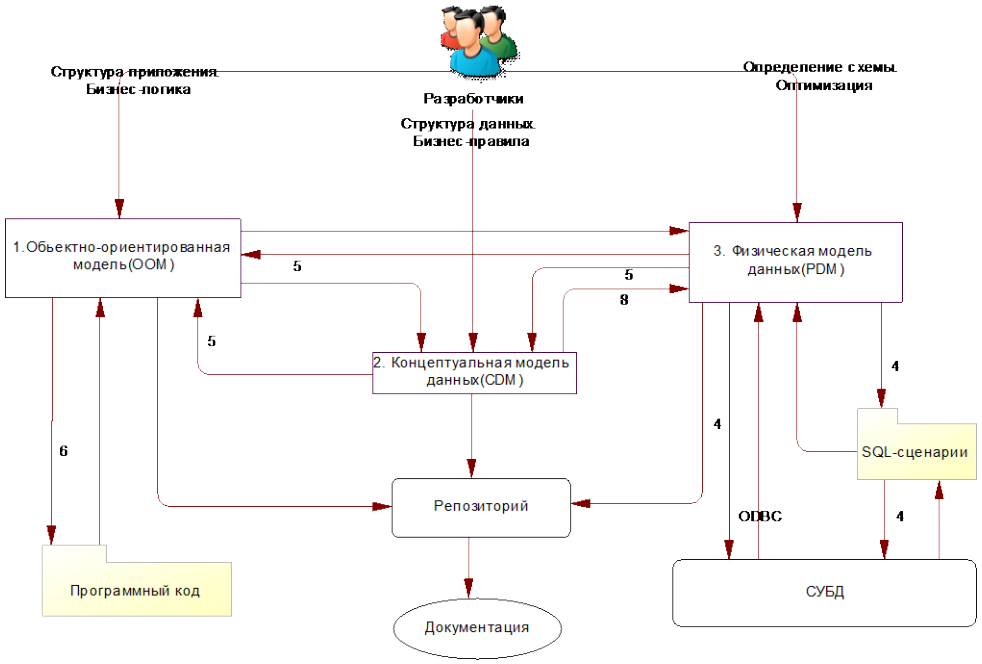


Рис. 1.49. Схема возможных преобразований моделей в Power Designer

Последовательность проектирования и разработки проекта описана в табл. 1.3. Номера операций соответствуют числам на схеме.

Таблица 1.3. Последовательность проектирования и разработки проекта

№	Исходные данные	Средство разработки	Описание этапа проектирования и результат
1	Требования к структуре приложения и функциональным возможностям; способы применения и пользователи	PowerDesigner / модуль OOM	Проектирование концептуальной модели приложения в нотации UML-диаграмм прецедентов (use-CASE), сценариев (sequence или scenario), классов (class) и пр. *Результат*: диаграмма OOM
2	Данные о предметной области проектируемого приложения	PowerDesigner / модуль CDM	Проектирование концептуальной модели данных в нотации IE: определение сущностей и установка связей между ними.

			Результат: диаграмма CDM
3	Концептуальная модель данных (CDM)	PowerDesigner / модуль PDM	Прямой инжиниринг CDM в физическую модель данных (PDM) и доработка последней (адаптация к выбранной СУБД, оптимизация). *Результат*: диаграмма PDM
4	Физическая модель данных (PDM)	PowerDesigner / модуль PDM, СУБД	Генерация сценариев DDL и передача их на СУБД (через ODBC или текстовый файл с запросами SQL). *Результат*: модель данных в конкретной СУБД
5	ОО и физическая модели данных (OOM и PDM)	PowerDesigner / модуль OOM	Объединение (merge) моделей, преобразование структур данных PDM в классы OOM, обновление CDM (при необходимости). *Результат:* дополненная диаграмма OOM
6	Объектно-ориентированная модель OOM	PowerDesigner / модуль OOM	Прямой инжиниринг OOM в исходные тексты выбранного средства разработки приложения. *Результат:* Шаблоны или готовый код на Java, PowerBuilder, C++, VB или другом языке программирования
7	Исходные тексты из п. 6 (см. рис. 1.49).	Выбранное средство разра-ботки (Sybase PowerBuilder, PowerJ, Sun TDK, Microsoft VC++)	Доработка приложения с помощью средств RAD или «обычного» программ-мирования (разработка пользовательского интерфейса, функции ввода/вывода и пр.). *Результат:* дополненный программный код и готовое исполняемое приложение (на конкретном витке итерации)
8	Исходные тексты из п. 7	PowerDesigner / модули OOM, CDM, PDM	Обратный инжиниринг обновленных классов в диаграммы UML модели OOM и обновление CDM, PDM, структуры БД (при необходимости). *Результат:* обновленные модели OOM, CDM, PDM

Характеристика Power Designer 16.0

Интерфейс

Расположение панели инструментов

Панель инструментов с функцией Auto Hide. Есть возможность фиксации этой панели к любому из четырех углов рабочей области, что увеличило удобство работы с инструментом.

Работа с открытыми диаграммами

Появилась возможность видеть список открытых диаграмм (рис. 1.50) в виде закладок в верхней части области отображения диаграмм, переключаться между ними и даже «вытащить» нужные диаграммы за пределы стандартной области редактирования, сделав их отдельным окном, и таким образом работать с несколькими диаграммами одновременно.

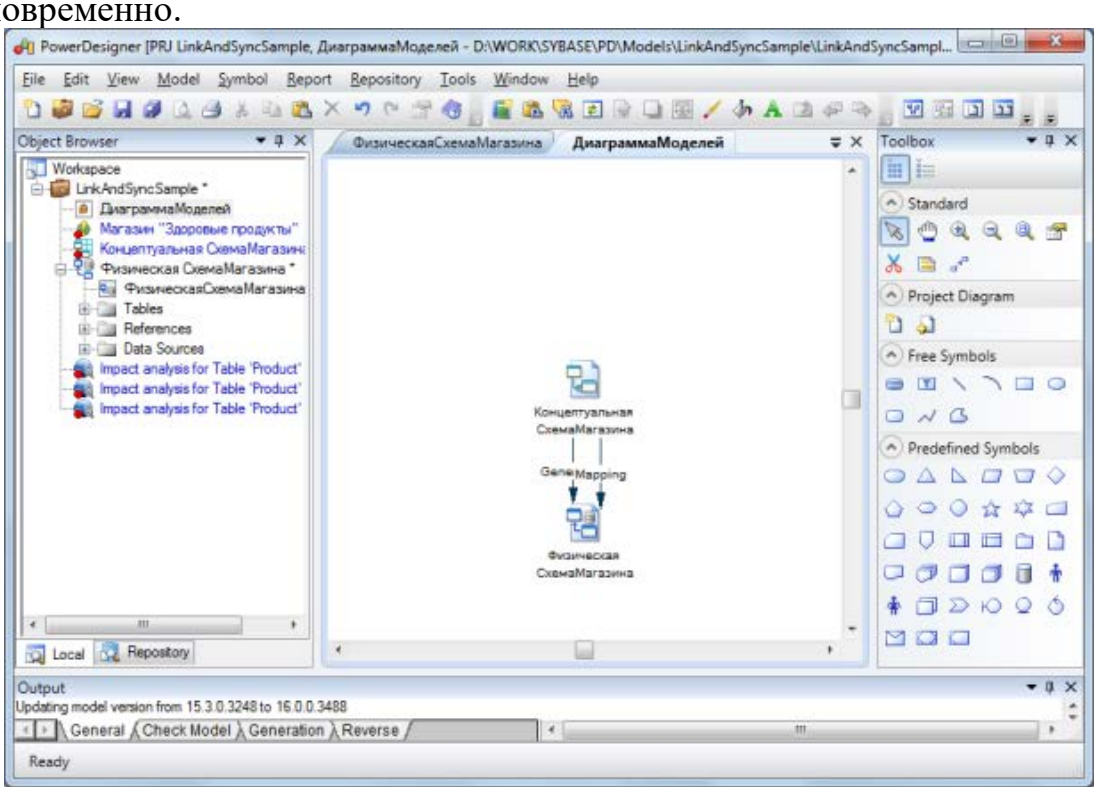


Рис. 1.50. Главное окно разработки моделей в Power Designer 16.0

Управление тубларами и их настройка

Третьим задекларированным новшеством является более гибкая работа с тулбарами, т. е. возможность зафиксировать их конфигурацию (Lock Toolbars), а также полностью настроить их содержимое через меню Customize Menus and Tools (здесь же можно настроить и содержимое приборной панели для каждого вида модели). Все это доступно через контекстное меню тулбаров.

Профили пользователей (User profiles)

Группы настроек:

Display Preferences – опции отображения объектов диаграммы (хранятся в модели и в registry). Контролируют цвет, внешний вид, размеры, состав и расположение отображаемой информации для различных символов диаграммы.

Model Options – опции модели (хранятся в модели). Контролируют соглашения о наименовании объектов, используемую нотацию, значения по умолчанию, чувствительность к регистру и т. п. Конкретный набор опций зависит от типа модели.

General Options – общие характеристики (хранятся в registry). Контролируют внешний вид и поведение интерфейса PowerDesigner, например, настройки диалогов, переменные окружения, шрифты и т. п.

Check Model options – опции проверки модели (хранятся в модели). Контролируют набор проверяемых параметров модели и уровень реакции на несоответствие (ошибка, предупреждение). Набор правил проверки зависит от типа модели.

Другие опции (хранятся в registry). Сюда относятся расположение тулбаров и окон (Organizing Views), любимые закладки для свойств объекта, набор отображаемых по умолчанию колонок для списков объектов и т. п.

Новшеством PowerDesigner 16.0 является тот факт, что теперь можно заранее задать различные наборы этих настроек и применять ту конфигурацию, которая более всего подходит для целей текущего проекта. Делается это через *Профили пользователей* или *User profiles*.

В меню Tools->Resourses->User Profiles есть список профилей по умолчанию. Можно просмотреть и отредактировать параметры каждого профиля или создать свой собственный профиль, взяв за основу один из имеющихся либо (что очень полезно) текущие настройки модели, либо информацию о настройках из системного реестра Windows (registry).

Применить созданный или входящий в поставку профиль можно через меню Tools->Apply User Profile. При этом следует иметь в виду, что если новый профиль применяется к модели, которая уже находится в разработке, то параметры отображения созданных диаграмм останутся неизменными. Новые настройки будут применяться только к новым диаграммам. Поэтому имеет смысл использовать нужный профиль в самом начале работы с моделью.

Работа с диаграммами

Расширенные возможности редактирования символов диаграммы

Увеличились возможности по редактированию содержимого символов на диаграмме. В особенности это касается символов, которые отображают сложные объекты, содержащие внутренние коллекции элементов (таблицы, сущности, классы и т. п.). Теперь можно отредактировать объект прямо на диаграмме без захода в список свойств объекта. Строка с выбранным элементом подсвечивается, можно перемещаться вверх и вниз по списку соответствующими клавишами, перетащить или скопировать элемент (например, колонку таблицы) из одного объекта в другой, изменить его местоположение в списке, выделить несколько элементов и т. п.

Горизонтальное и вертикальное расположение элементов на символе диаграммы

Расширилась возможность настройки отображаемого содержимого объекта на диаграмме. Теперь можно не только задать любые атрибуты и

коллекции объекта, которые нужно отобразить, но и описать их местоположение. Сущность в концептуальной модели данных показана на рис. 1.51.

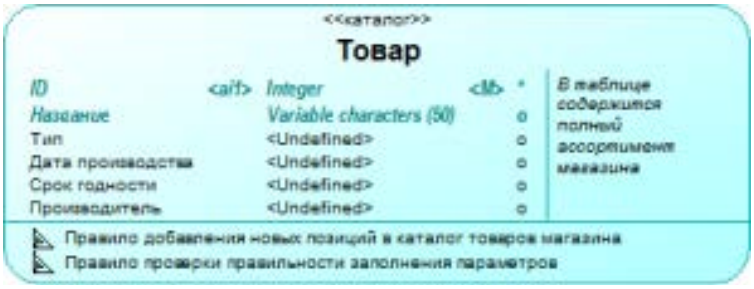


Рис. 1.51. Сущность в концептуальной модели данных

Выделены отдельным шрифтом заголовок сущности и два ее основных атрибута. Комментарий расположен справа от атрибутов. Выбор и расположение объектов было задано через меню Tools->Display Preferences, объект Entity. На закладке Content для этого объекта есть кнопка Advanced…, открывающая диалог Customize Content. Настройки диалога для создания сущности «Товар» можно увидеть на рис. 1.52.

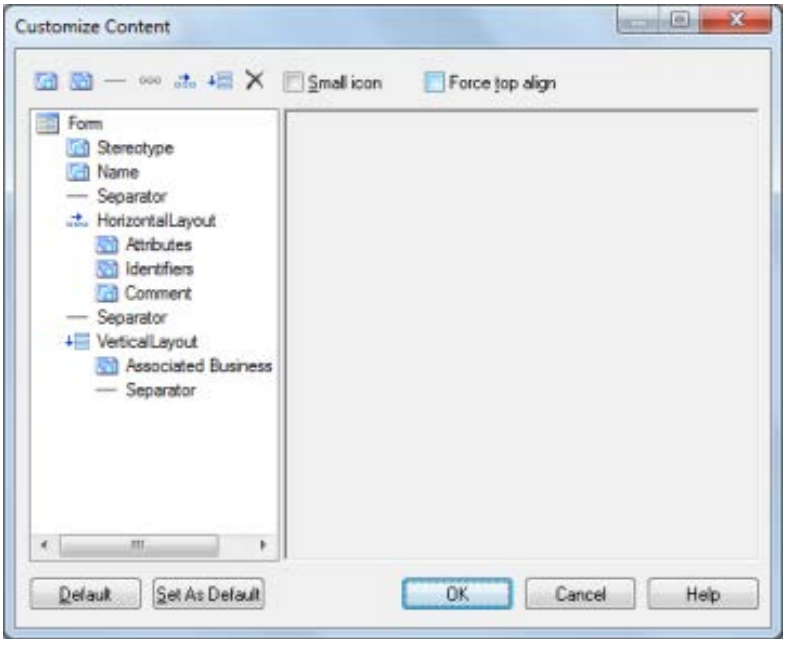


Рис. 1.52. Диалог Customize Content

Новые возможности

Отображение всех свойств объекта-ссылки (*shortcut*)

В PowerDesigner существует возможность переноса объектов из одной модели в другую. Один из способов – воспользоваться механизмом ссылки (shortcut). Это позволяет создавать целые библиотеки стандартных объектов для того, чтобы использовать их впоследствии в других моделях.

Рис. 1.53. Реализация объект-ссылка

Объект-ссылка на объект, физически расположенный в модели «PD16_CDM» (пиктограмма в левом нижнем углу) изображен на рис. 1.53 слева. При этом модель «PD16_CDM» в данный момент закрыта. Пример физической модели данных (на базе ссылка на часть 1. рис. 2.20) показан на рис. 1.53а.

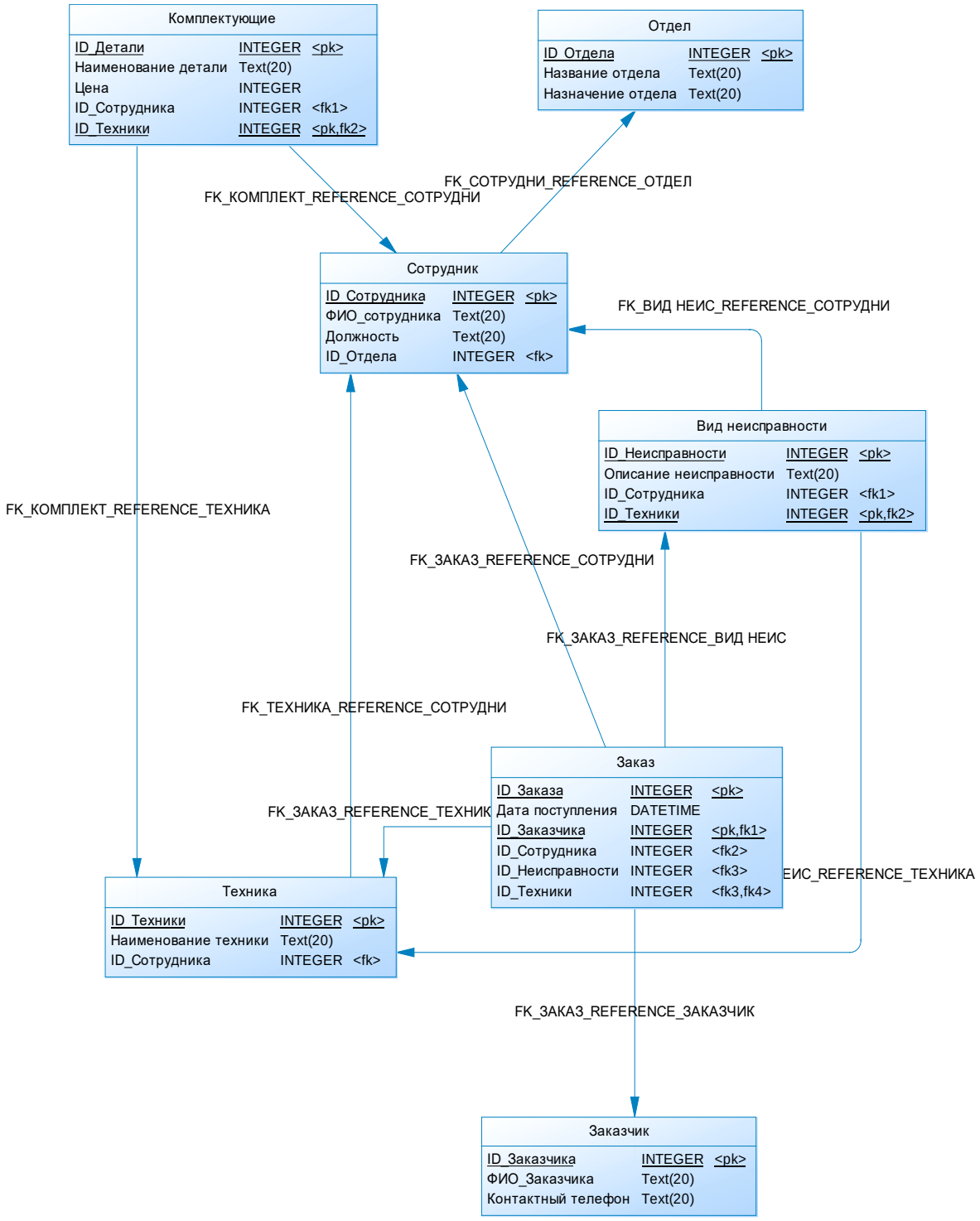


Рис. 1.53а. Реализация физической модели данных

Увеличено удобство работы с дополнительными связями объекта

В предыдущих версиях Power Designer существовало понятие Extended Dependency – особый вид связи, который позволял связать объект модели с любым другим объектом той же самой или любой другой модели. Эти связи использовались для описания семантических связей между объектами самых разных моделей. Например, можно было связать бизнес-процесс с таблицей в базе данных, а XML-схему – с каким-либо

UML-объектом из объектно-ориентированной модели. Существовала возможность просмотреть все связи такого рода для заданного объекта, – они отражались на закладке Extended Dependency в свойствах объекта. Этот механизм сам по себе уже был достаточно удобен, однако в версии PowerDesigner 16.0 возможности работы с подобными связями существенно расширены.

Теперь Extended dependencies носят название Traceability links (трассируемые ссылки). Также, как и раньше, их создают для связи между объектами любого типа. Новшеством является то, что теперь можно проводить типизацию этих связей, а затем группировать их по классу связываемых объектов или по типу. Трассируемые ссылки объекта доступны на закладке Traceability Links, заменившей закладку Extended dependencies из предыдущих версий (рис. 1.54). Здесь отображены все созданные трассируемые ссылки для объекта «Склад» из концептуальной модели данных. При создании этих связей объявляются два типа ссылок: «Процессы» и «Территории». Ссылки первого типа указывают на процессы из моделей бизнес-процессов, в которых участвует объект, ссылки второго типа – на объекты типа Site из инфраструктурной диаграммы модели архитектуры предприятия. По умолчанию все имеющиеся ссылки никак не группируются. Закладка с группировками изображена на рис. 1.55.

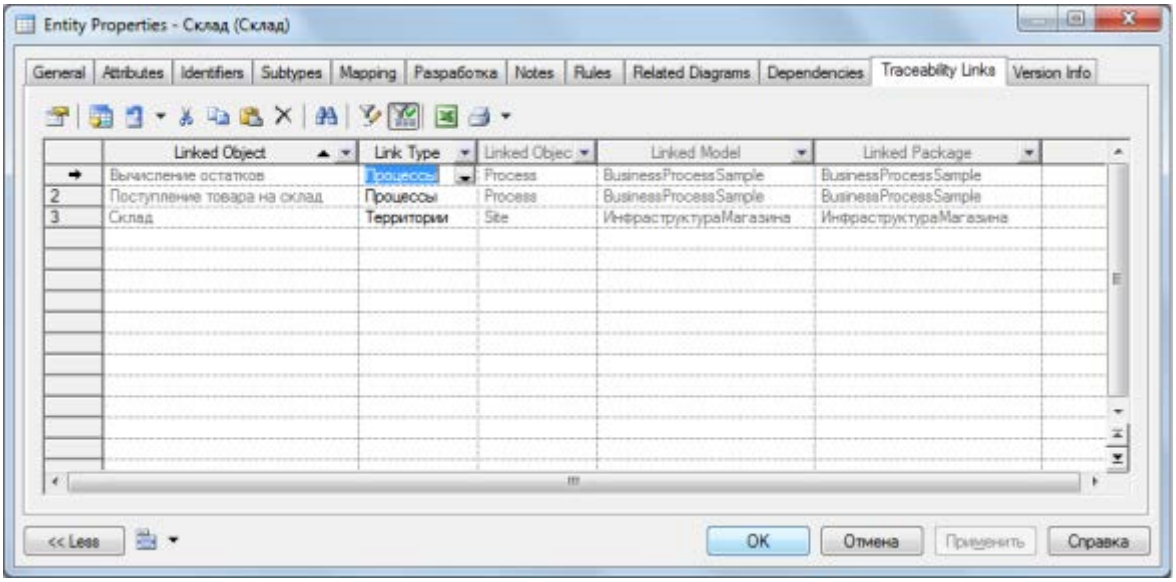


Рис. 1.54. Трассируемые ссылки объекта

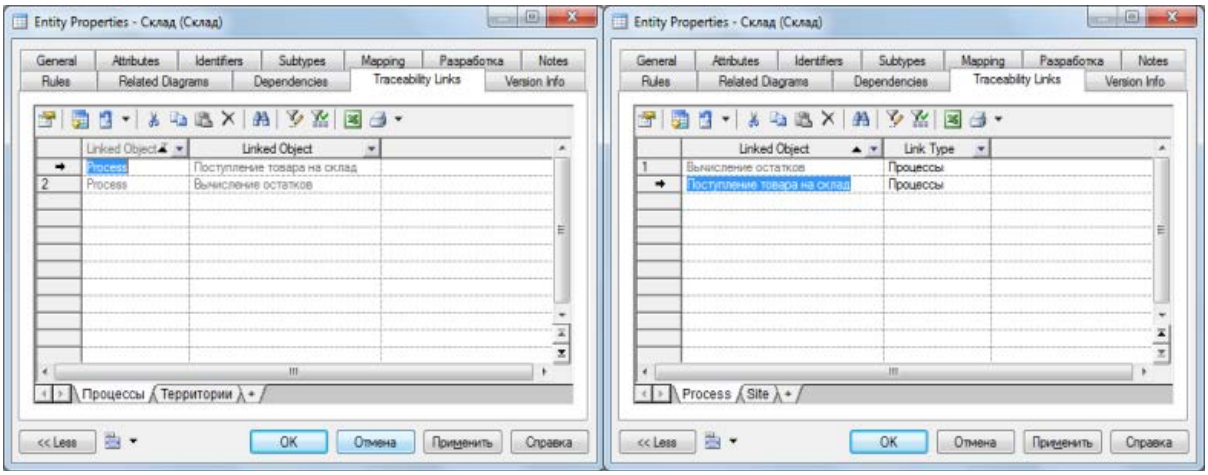


Рис. 1.55. Сгруппированные ссылки

Слева ссылки сгруппированы по типу ссылки (переключаться между списками объектов для ссылок разных типов можно при помощи закладок внизу), справа – по типу объекта.

Поддержка работы в новых средах

Теперь поддерживаются: версия Windows 64-bit, интеграция с Eclipse v3.6, интеграция с Microsoft Office 2010 и импорт из Visio 2010.

Корпоративная библиотека

Представим себе ситуацию в крупной компании, где имеется множество различных отделов, каждый из которых занимается разработкой своих проектов. Чтобы эти проекты были согласованы друг с другом и в потенциале могли достаточно легко интегрироваться, необходимо введение стандартизации. Обычно это имена и названия объектов, использование одних и тех же типов данных в БД или некоторого набора общих объектов и т. п. Нужно выделить общие объекты в некоторый набор библиотечных моделей, чтобы использовать их в различных проектах. Для этих целей и служит корпоративная библиотека в Power Designer 16.

Библиотека – это специально выделенный раздел репозитория, в котором администратор (или другой пользователь, обладающий соответствующими правами) размещает модели и любые другие документы общего пользования. Пользователи могут использовать объекты библиотечных моделей в своих моделях, например, в качестве объектов-ссылок (shortcut) или реплик (replica).

При соединении с репозиторием перед пользователем появляется окно, в котором содержится список объектов библиотеки, где он может выбрать, какие из них загрузить в локальное рабочее пространство (рис. 1.56).

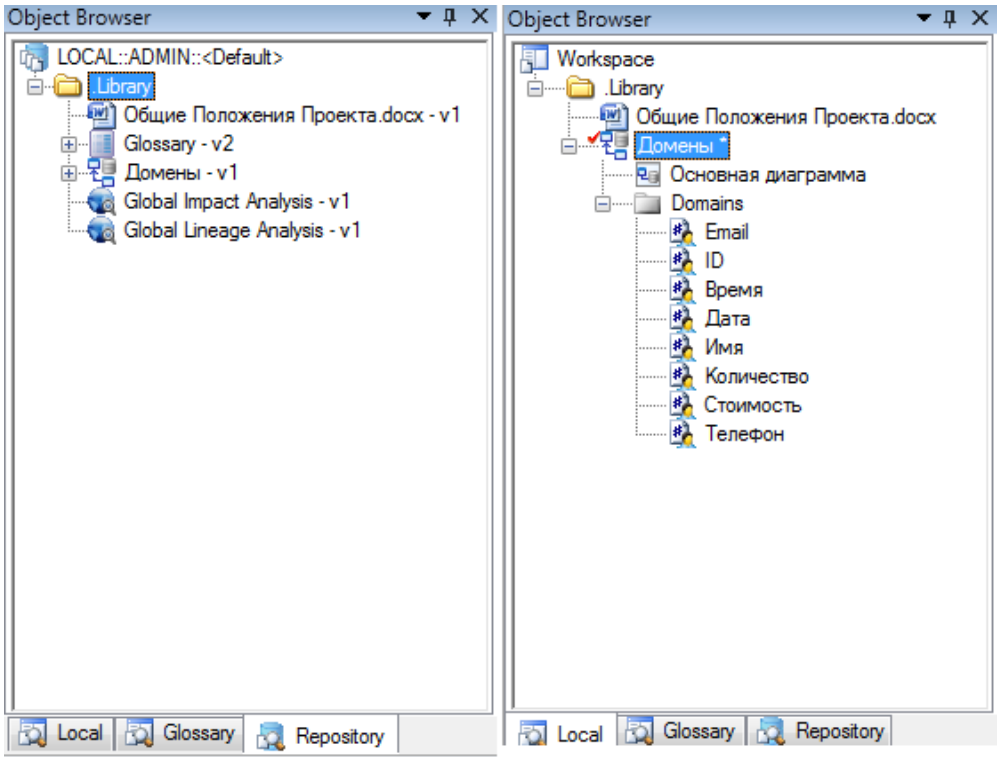


Рис. 1.56. Список объектов библиотеки

Слева изображено содержимое библиотеки в репозитории, справа – содержимое локальной библиотеки пользователя.

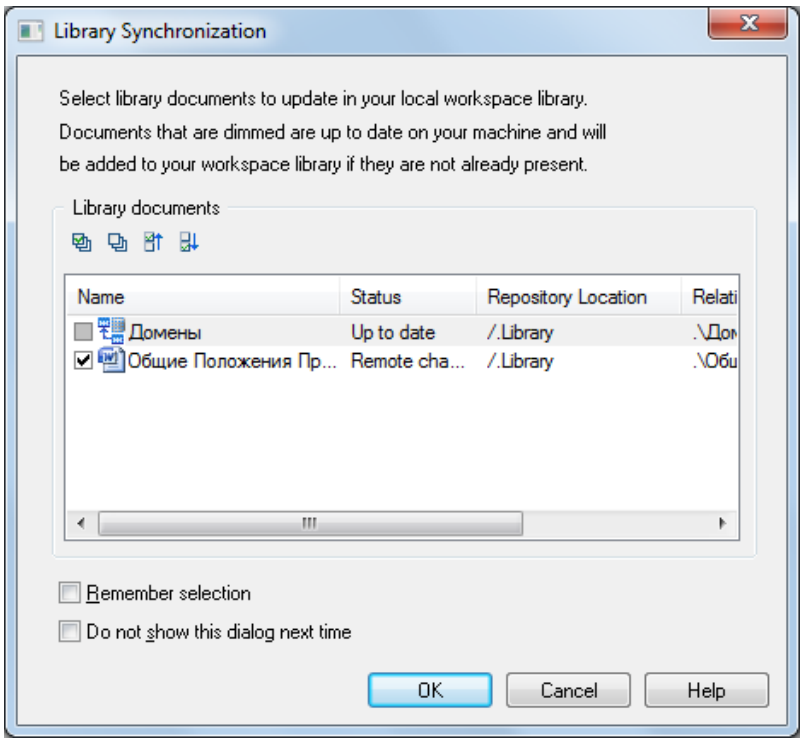


Рис. 1.57. Окно синхронизации библиотеки

Окно синхронизации библиотеки, появляющееся при соединении с репозиторием, представлено на рис 1.57.

Корпоративный глоссарий

Эта функция полезна для крупных компаний, в которых ведется параллельная разработка и развитие большого числа проектов и/или баз данных, взаимодействующих друг с другом. Задача корпоративного глоссария – стандартизировать имена и коды объектов во всех моделях. Если PowerDesigner используется совместно с репозиторием, администратор может создать необходимый глоссарий и поместить его в библиотеку репозитория. Пользователи автоматически получают обновления глоссария каждый раз при соединении с репозиторием. Далее, при создании нового объекта, когда пользователь начинает вводить его имя, в выпадающем списке предлагается выбрать подходящие наименования из глоссария. Также при проверке модели возникает сообщение об ошибке, если какие-либо объекты имеют названия, отсутствующие в глоссарии.

Заполнение глоссария производится на закладке Glossary браузера, которая появляется при наличии настроенного доступа к репозиторию и соответствующих прав. Для удобства термины глоссария можно разделить на категории, уровень вложенности которых не ограничивается. Первичное заполнение глоссария можно выполнить путем импорта наименований из имеющейся модели или из Excel файла (рис. 1.58).

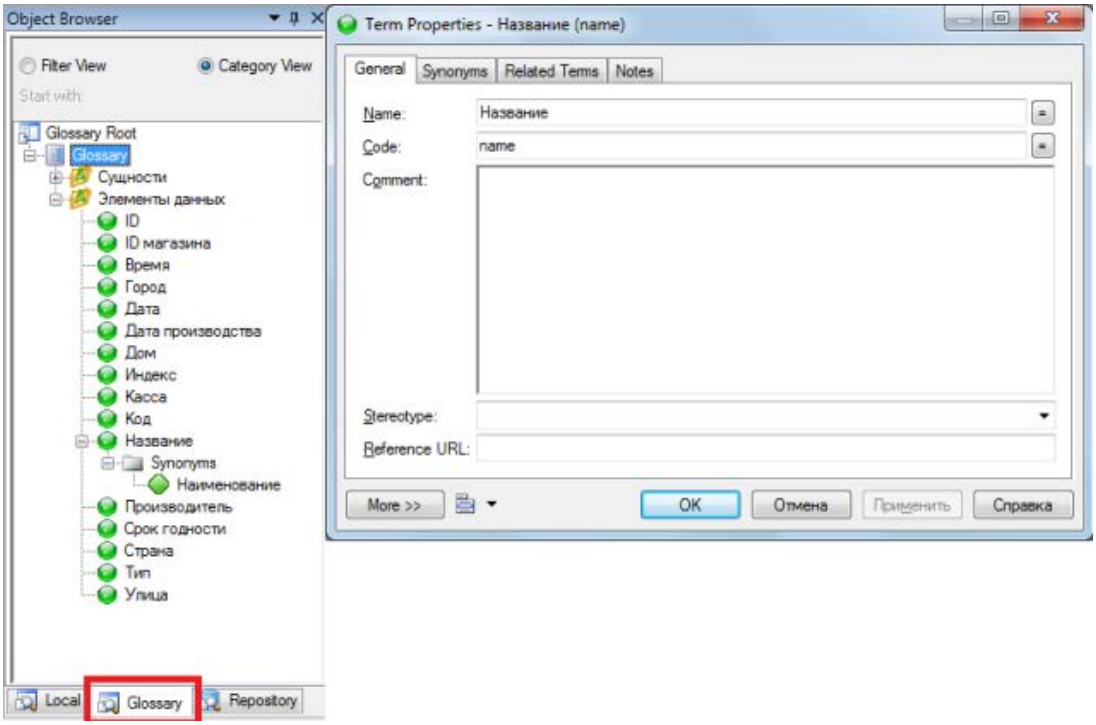


Рис. 1.58. Заполнение глоссария

Слева изображено дерево элементов глоссария в разбивке по категориям, справа – карточка свойств одного из элементов.

При создании новой сущности в модели данных при вводе наименования объекта Power Designer выдает подсказку с подходящими именами из глоссария (рис. 1.59).

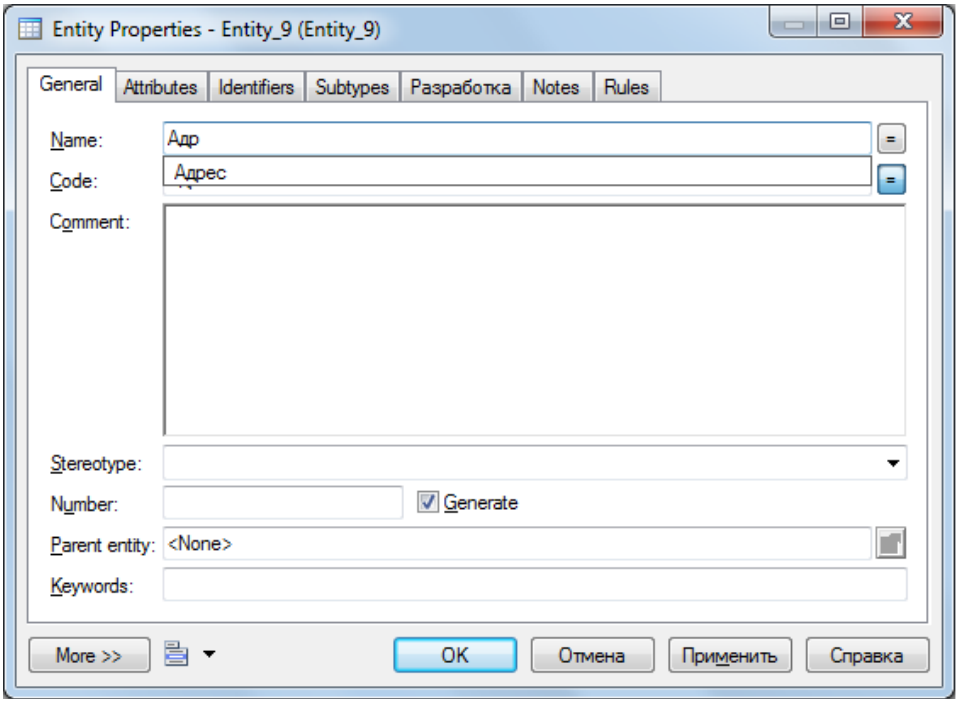


Рис. 1.59. Подсказка с подходящими именами из глоссария

Настройка интерфейса Power Designer для пользовательских ролей

В крупной компании, как правило, разные группы пользователей используют Power Designer для различных задач: одна команда пишет требования, вторая – прорабатывает бизнес-процессы, третья – модели данных, четвертая – продумывает архитектуру приложения и, наконец, пятая – архитектуру предприятия в целом. Power Designer способен удовлетворить все эти потребности моделирования, но большое число функций, опций и возможностей может усложнить процесс освоения и работы с инструментом отдельного пользователя. Кроме того, у каждой из этих команд своя область ответственности: будет неправильно, если процесс, создаваемый разработчиком бизнес-процессов, начнет править кто-то из команды по моделированию данных, но разработчику модели необходимо иметь доступ для просмотра модели процессов.

В Power Designer 16 появилась возможность создания интерфейсных профилей, позволяющих ограничить возможности пользователя так, как этого требуют корпоративные правила, и скрыть лишние пункты меню. Поскольку информация о профилях хранится в репозитории, для их

создания и редактирования требуется наличие соединения с репозиторием и права администратора.

Типы профилей:

1. *Профиль доступа к объектам* (*Object permissions profiles*) – предназначен для регулирования доступа к моделям различных типов, к их объектам и свойствам объектов. Позволяет скрыть от пользователя модели различных типов или сделать их доступными только для чтения. Здесь же есть возможность скрыть объекты моделей и их свойства. Используется, например, для того, чтобы упростить среду моделирования для различных групп пользователей, показав только те модели, которые нужны им для работы.

Настройка профиля осуществляется через меню Repository->Administration->Object Permissions profiles. После создания нового профиля в дерево слева (рис. 1.60) добавляются типы моделей, параметры доступа к которым нужно задать. Затем для каждой модели настраиваются параметры доступа: полный доступ (Enable), доступно только чтения (Read-Only), скрыто (Disable).

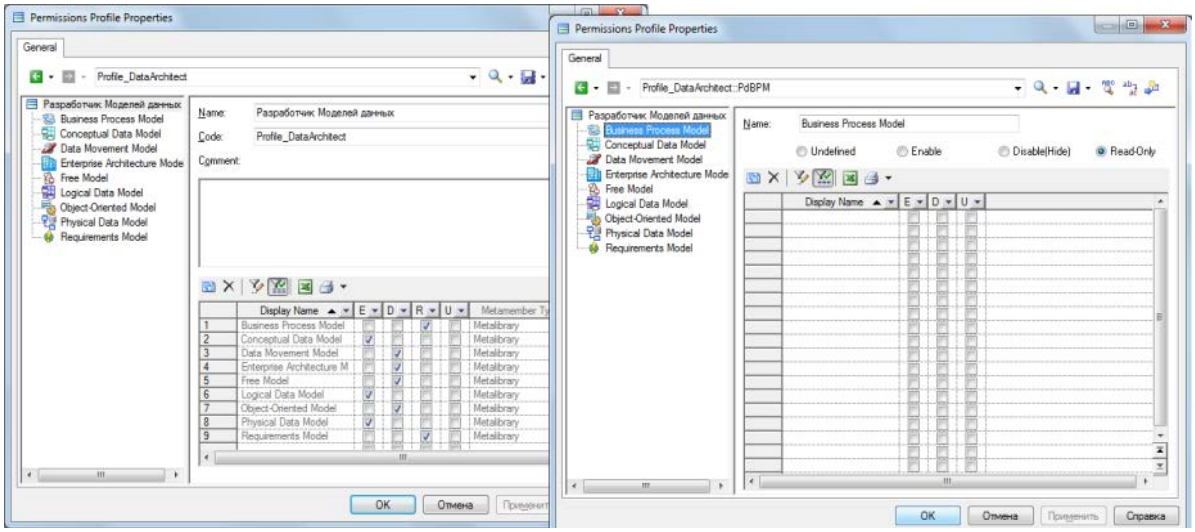


Рис. 1.60. Настройка профиля доступа к объектам

Слева отображен список моделей профиля «Разработчик моделей данных». Для моделей типов Conceptual Data Model, Logical Data Model и Physical Data Model задан полный уровень доступа (Enable), модели типа Business Process Model и Requirements Model доступны только для чтения (Read-Only), а остальные скрыты (Disable).

Аналогичным образом для модели каждого типа можно скрыть любой из относящихся к ней объектов (рис. 1.61).

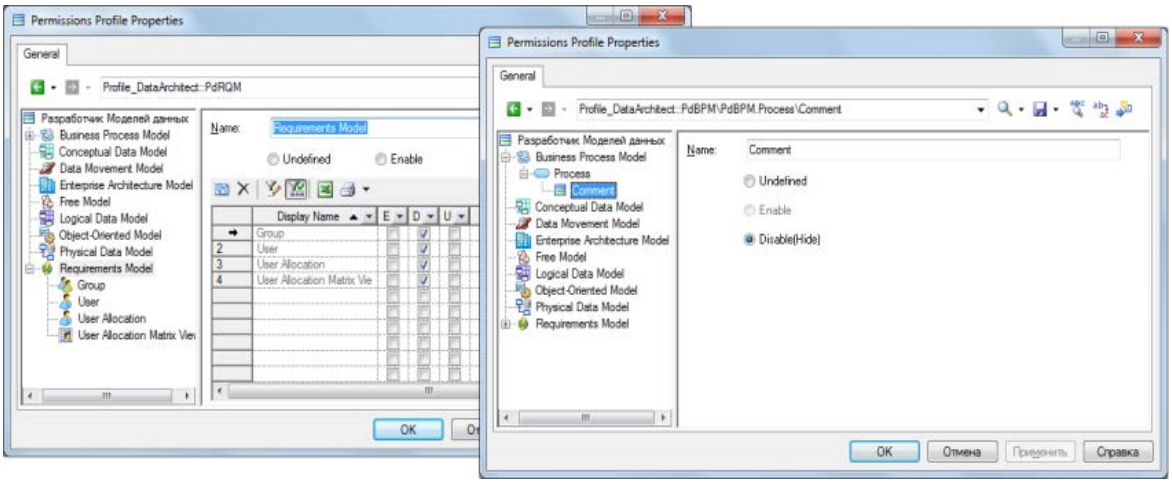


Рис. 1.61. Скрытие объектов

Слева выключены все объекты, относящиеся к пользователям и их группам, а справа – свойство Comment в объекте Process модели бизнес-процессов.

2. *Профиль настроек пользовательского интерфейса* (*UI preferences profile*) – позволяет настроить видимые опции меню, тулбаров и инструментальных панелей. Фактически здесь задаются те же опции, что и через меню Tools -> Customize Menus and Tools. Призван повысить удобство работы пользователя с интерфейсом, скрыв ненужные функции. При этом, в отличие от профиля предыдущего типа, он задает лишь опции по умолчанию. Пользователь может поменять эти настройки в своем локальном рабочем пространстве.

3. *Профиль общих настроек* (*General preferences profile*) – предназначен для установки опций отображения (Display Preferences), опций модели (Model Options) и общих настроек PowerDesigner (General Options). Как и в предыдущем случае, здесь задаются только опции по умолчанию, которые впоследствии могут быть изменены пользователем. Настройки этого профиля аналогичны настройкам, которые пользователь может создать для себя персонально через интерфейс User Profiles (рис. 1.62). Созданные профили можно связать с конкретным пользователем или группой пользователей на закладке Profiles в его настройках.

Анализ взаимосвязей объектов по моделям репозитория

Новшеством Power Designer.16 является тот факт, что анализ связи между различными объектами разных моделей можно распространить и на объекты, находящиеся в репозитории и не открытые в локальном рабочем пространстве пользователя. Раньше для осуществления полного анализа по всем моделям было необходимо, чтобы все они были открыты.

Результат анализа влияний таблицы «Product» из физической модели данных изображен на рис. 1.62. В результате анализа объекты из репозитория помечены пиктограммой в левом нижнем углу объекта.

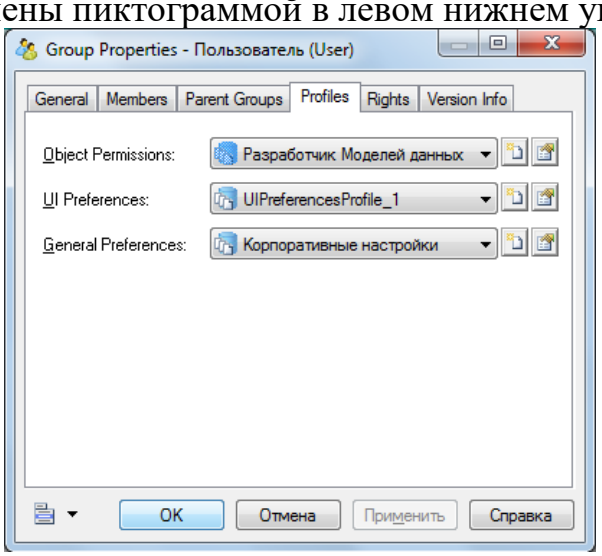


Рис. 1.62. Профиль общих настроек

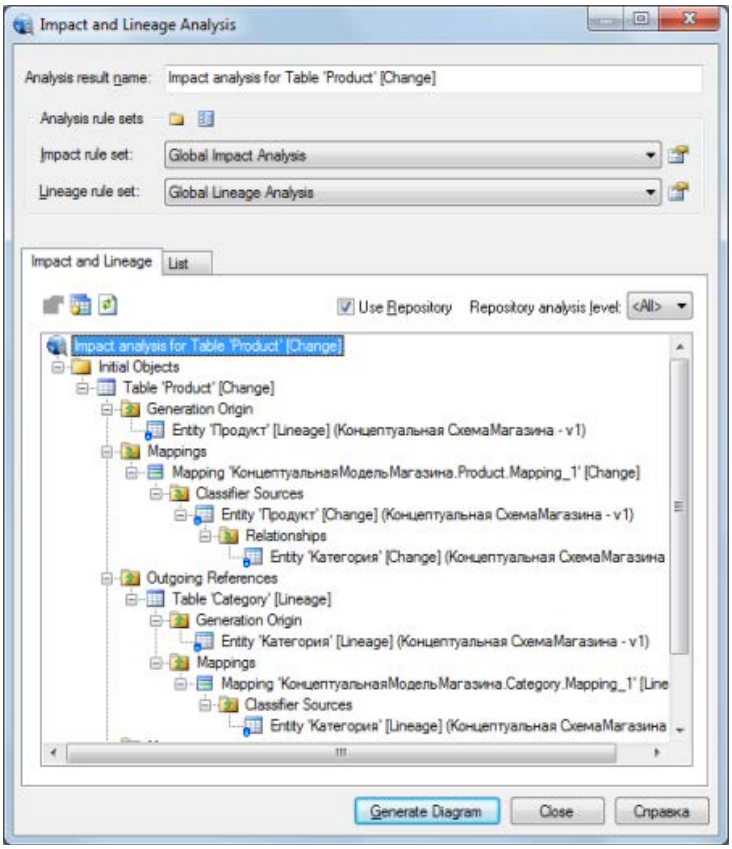


Рис. 1.63. Результат анализа связей объектов

Взамосвязь Power Designer с другими инструментами позволяет расширить возможности разработчиков программных продуктов.

ГЛАВА 2. ПРИМЕРЫ МОДЕЛЕЙ ПРЕДМЕТНЫХ ОБЛАСТЕЙ АВТОМАТИЗАЦИИ

2.1. Информационная система «Телефонная служба приема заявок»

В качестве примера рассмотрим систему «Телефонная служба приема заявок» и приведем фрагменты этой системы, изображенные с помощью различных UML-диаграмм. Примеры будут снабжены некоторыми «сюжетами» – гипотетическими ситуациями процесса разработки, в которых могла появиться необходимость в создании этих диаграмм. Разумеется, приводимые сюжеты далеко не единственные, даже в рамках разработки данной системы. Но они нужны, чтобы с первых же шагов при знакомстве с UML не появлялось чувство незавершенности иллюстраций.

Часть примеров будет на русском языке, а часть – на английском (это касается всех дальнейших примеров, а не только тех, которые относятся к телефонной службе приема заявок). Если диаграммы связаны с программным кодом, т. е. моделируют какие-либо его абстракции (классы, таблицы баз данных, компоненты и пр.), то используется англоязычная терминология – названия модельных сущностей должны быть идентификаторами в программном коде. Если же такого нет, то при именовании используется обычный русский язык.

Итак, заказчик системы – это компания, владеющая сетью продуктовых магазинов, которая кроме обычной розничной торговли планирует предоставлять сервис по обслуживанию клиентов по телефонным заявкам. Иными словами, клиент регистрируется в компании, а потом по телефону в удобное для себя время делает заказ товаров и расплачивается при получении. Для этого компании нужно организовать у себя локальный телефонный центр, состоящий из офисной многоканальной автоматической телефонной станции (АТС), штата операторов и соответствующего программного обеспечения. При этом в компании уже есть информационная система по обработке заявок от постоянных мелкооптовых клиентов, и заказываемая система должна быть с ней проинтегрирована.

Диаграммы вариантов использования (use case diagrams)

Первым шагом по реализации описанной выше задачи является уточнение требований. Для этого можно применить *диаграммы вариантов* (*случаев*) использования UML. Одна из таких диаграмм показана на рис.2.1. На ней обозначены следующие виды пользователей: оператор, менеджер и представители технической поддержки. Кроме того, система должна поддерживать внешний *интерфейс* с системой обработки заявок. Это – четвертый пользователь. Еще одним пользователем системы является директор департамента сбыта товаров, который периодически

отслеживает деятельность телефонной службы приема заявок (Петров А.Б.). Для него создано специальное пользовательское место с экранными формами статистики.

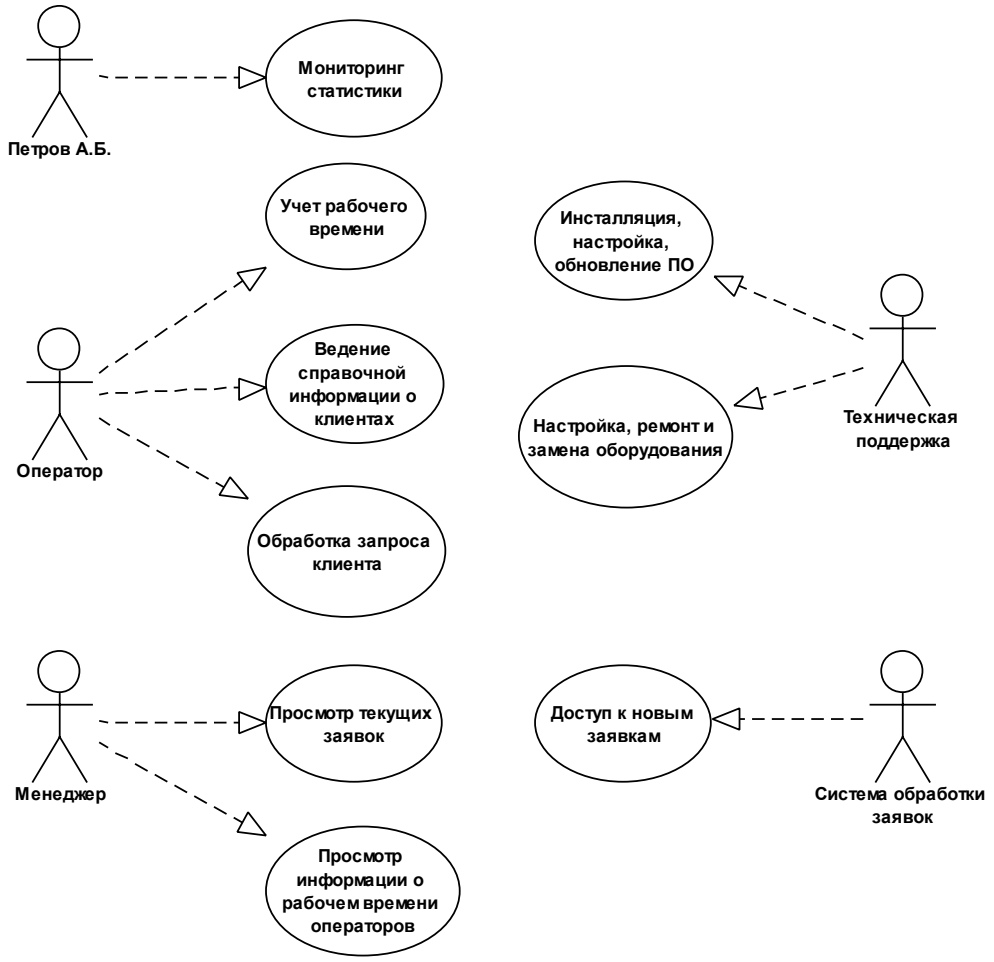


Рис. 2.1. Диаграмма случаев использования

Различные пользователи ПО, изображаемые на *диаграммах случаев* использования, называются *актерами* (actors). Они могут обозначать:

типовых пользователей («Менеджер», «Оператор», «Техническая поддержка») – работников компании, сгруппированных по исполняемым обязанностям;

другие системы, взаимодействующие с указанной («Система обработки заявок»);

выделенного пользователя («Петров А.Б.»).

Отметим, что выделенный пользователь существенно отличается от типового. Он, как правило, имеет особый приоритет, и функциональность для него обязательно согласуется с ним. Часто он влияет на оплату проекта, от его мнения о системе во многом зависит ее успешная сдача.

Случай использования (use case) – это независимая часть функциональности системы, обладающая результирующей ценностью для ее пользователей.

«Независимость» означает, что если случай использования всегда исполняется вместе с некоторым другим, то, по всей видимости, один из них нужно включить в другой (как назвать получившийся в итоге случай использования – зависит от обстоятельств).

«Результирующая ценность» случая использования для актера системы подразумевает, что данный *случай использования,* должен приносить *актеру* некоторый законченный и ценный с точки зрения его бизнеса результат. Будучи реализован системой, этот случай использования действительно делает бизнес актера эффективнее, производительнее. Тем самым разработка системы фокусируется на бизнес-целях, а незначительные случаи использования игнорируются. Строится не абстрактная модель функций системы, а набор самых важных (для заказчика и пользователей) сервисов, чтобы каждый из них правильно понять и не один не упустить. В дальнейшем контроль разработки системы будет осуществляться именно в терминах самого важного – того, что нужно заказчику и пользователям.

Реализовать набор функций, необходимых пользователю, непросто, так как на практике программный проект может незаметно потерять эту цель. Можно, например, очень долго заниматься разработкой сложной и многофункциональной архитектуры, после реализации которой разработчики обещают, что все пользовательские функции будут правильно работать. Однако на практике оказывается, что проект выбивается из расписания, а многие заказанные пользователем функции сделать тяжело или невозможно. Таким образом, чрезмерная ориентация на «внутреннее совершенство» ПО оканчивается для проекта либо крупными неприятностями, либо полным крахом. Однако бывают и другие случаи, когда только такая ориентация впоследствии и спасает проект. Последнее случается, если система долго развивается и сопровождается, или когда требования к ней внезапно и сильно меняются, или когда на ее основе делаются другие системы. Необходим баланс между внутренним совершенством программного обеспечения и функциональностью, нужной для заказчика и доставленной ему в срок. Разработка в терминах случаев использования – хороший способ контролировать, что процесс создания системы двигается в нужном направлении.

Основной задачей *диаграмм случаев использования* является получение требований к системе от заказчика и пользователей. Трудность формализации требований связана с тем, что, с ооодддной пользователи и заказчики с одной стороны, а программисты – с другой, являются специалистами в совершенно разных областях. Первым сложно понять логику программной разработки и отделить существенное от

несущественного, а вторым – трудно разобраться в новой предметной области и адекватно отразить свое понимание в программной системе. Программные системы очень часто являются уникальными, поэтому набор пожеланий заказчика и пользователей нуждается в дополнительной обработке, освобождению от противоречий, коррекции и, наконец, интеграции, так как стало возможным «покрыть» его некоторой программной системой. Привлекательность и эффективность диаграмм случаев использования заключается в простоте понимания непрограммистами и достаточной формальности.

Отметим, что сами по себе случаи использования не гарантируют того, что программисты и заказчик адекватно понимают друг друга – они могут по-разному трактовать эти случаи использования. Однако в первом приближении масштаб и границы системы очерчены. Для детализации случаев использования может применяться обычный текст (по одному абзацу на каждый случай использования) и/или другие диаграммы UML.

Существуют два вида принципиально разных *диаграмм случаев использования* – для ПО и для всей системы в целом. Ведь, как правило, ПО является частью более крупной системы. Последняя может включать другое ПО, а также некоторый бизнес-процесс. Пользователями такой системы будут различные клиенты (*бизнес-актеры*), а сама система будет предоставлять для них бизнес-случаи использования. Пример диаграммы бизнес-случаев использования для системы обработки телефонных заявок показан на рис. 2.2.

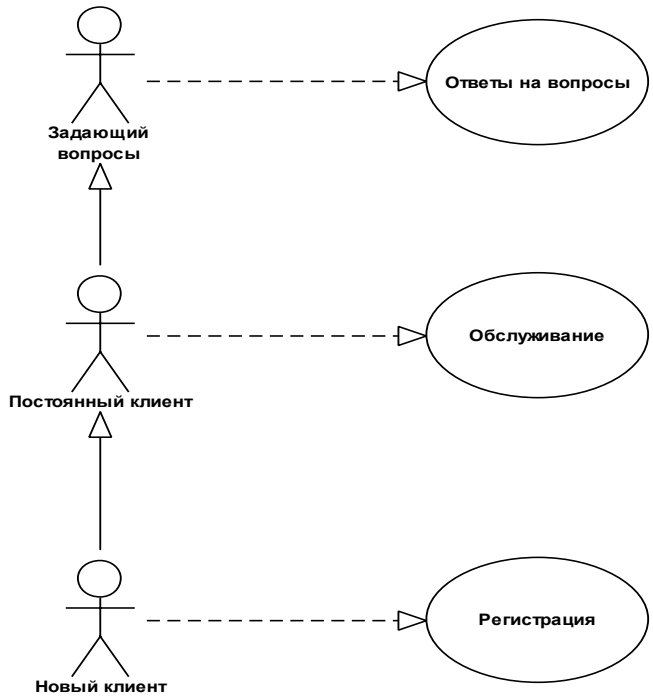


Рис. 2.2. Диаграмма бизнес-случаев использования

Можно увидеть трех различных клиентов этой системы – постоянного клиента, нового клиента и случайного человека, интересующегося услугами магазина и наличием того или иного товара. В общем, для каждого типа клиентов система должна предоставлять разный сервис: для первого – возможность сделать заказ (с внесением в базу данных имени клиента, заказанного товара, его цены и сроков доставки), для второго – возможность зарегистрироваться (оператор спрашивает у него фамилию, имя, отчество, адрес и т. д. вносит ее в компьютер), для третьего – возможность отвечать на разные вопросы (возможно, со специальными справочниками товаров и пр.). Причем эти *актеры* наследуют один от другого именно в том порядке, который указан на диаграмме. При наследовании актеров потомок «получает в наследство» все *случаи использования* своих «предков». Таким образом, каждый из этих клиентов может задавать вопросы, а новый клиент, после того как зарегистрировался, – сделать заказ.

Этих бизнес-клиентов можно было бы изобразить и на рис. 2.1, соединив стрелками с оператором (ведь именно через него они взаимодействуют с системой). Но такая диаграмма может вызвать недоумение, хотя некоторые аналитики склонны так делать. Бизнес-актеров рекомендуется изображать на отдельной диаграмме, а на обычной диаграмме случаев использования показывать только пользователей ПО.

Диаграммы активностей (*activity diagrams*)

Используются для изображения бизнес-процессов – алгоритмов, по которым работает компания. Именно в эти алгоритмы должна встроиться информационная система, автоматизировав некоторую их часть. В данном случае в компании должен быть создан новый бизнес-процесс по телефонной обработке заявок. Заказчик первоначально представляет будущий процесс, и перед началом разработки системы необходимо уточнить алгоритм работы новой службы. Общая схема работы оператора с клиентом показана на рис. 2.3.

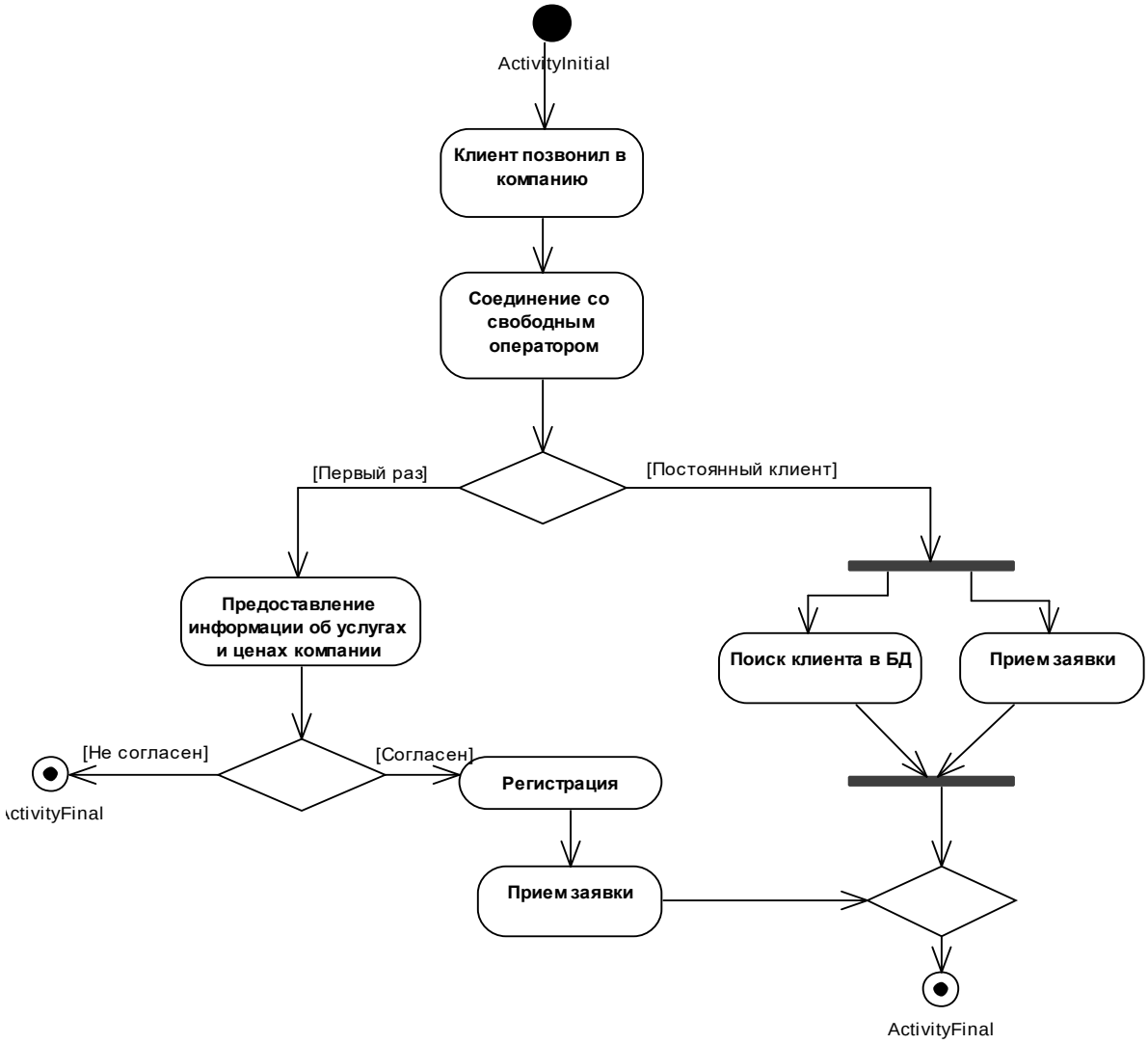


Рис. 2.3. Диаграмма активностей

Программистам полезно ясно представлять себе все бизнес-процессы компании, которые будут затронуты их новой системой. В данном случае у компании еще есть бизнес-процесс обработки заявок, который уже работает и есть у заказчика, и его также нужно понять. Иначе может оказаться, что упущена важная деталь, которая не позволяет новой системе полноценно выполнять свои функции. Например, подсистема обработки заявок, с которой должна интегрироваться создаваемая система, реализована на макросах к Word/Excel, что затрудняет интеграцию. На этот и подобные факты необходимо указать заказчику как можно раньше, так как иначе проект может закончиться неуспешно – заказчик потратит деньги и не получит нужных для своего бизнеса сервисов.

Итак, главной сущностью этого типа диаграмм является *активность* (activity) – активное состояние системы, в котором она выполняет некоторую работу. После ее завершения происходит переход в другую

активность. Возможны и более сложные случаи переходов между активностями, например, переход по событию.

На диаграмме должны присутствовать символы начала (start) и конца (finish).

Далее на диаграмме используется параллельный разветвитель (fork), который запускает несколько одновременно работающих веток. Такие ветки могут объединяться (все или только часть) конструкцией под названием *параллельный соединитель* (join).

Наконец, на диаграмме могут использоваться символы логического ветвления и логического соединения (decision). На ветках, идущих из логического ветвления, обозначаются условия перехода.

Диаграммы развертывания (*deployment diagrams*)

Теперь настало время в первом приближении определить будущую систему изнутри. Начнем с *диаграмм развертывания*, которые предназначены для описания аппаратной части системы.

Телефонная служба приема заявок, состоящая из офисной телефонной станции (*PBX* – Public Branch Exchange), сервера, телефонных аппаратов и клиентских компьютеров, показана на рис. 2.4а. Это диаграмма развертывания в описательном виде, на которой определены типы аппаратных узлов системы, а между ними – ассоциации с пометками множественности (см. описание диаграмм классов).

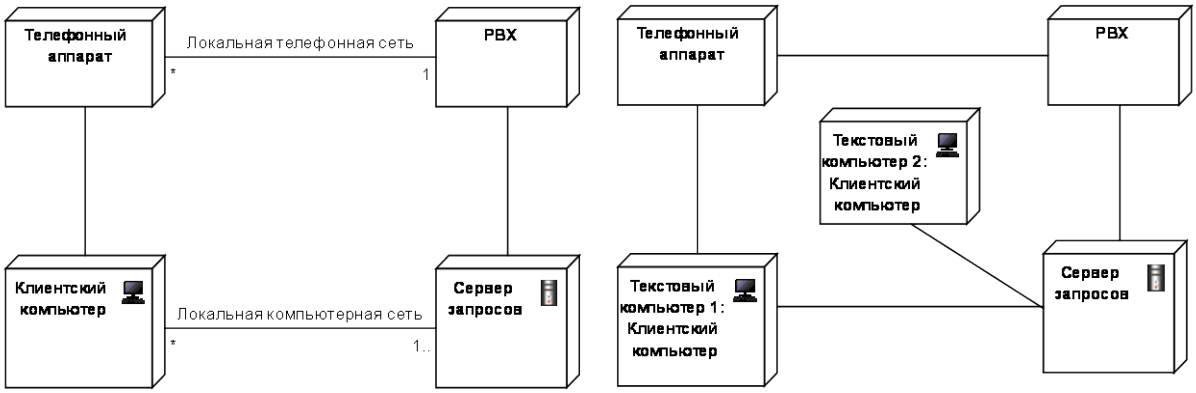
Диаграмма развертывания в экземплярном варианте приведена на рис. 2.4б. Показан тестовый вариант системы, который кроме сервера и *PBX* содержит один пользовательский компьютер для тестирования взаимодействия сервера и клиента и один клиентский компьютер вместе с телефонным аппаратом для тестирования связи клиента с сервером и *PBX*. Два клиентских компьютера нужны, чтобы тестировать работу ПО в случае более чем одного клиента (при переходе от одного к двум начинают появляться многочисленные ситуации, которые не проявлялись ранее). Большее количество клиентов – три, десять и т. д. – не принципиально на первых стадиях тестирования и отладки.

Описательный и экземплярный виды диаграмм развертывания соотносятся между собой так же, как *диаграммы классов* и *диаграммы объектов*.

Таким образом, на *диаграммах развертывания* показываются *узлы* (nodes) – элементы аппаратуры, которые также входят в целевую систему наравне с программным обеспечением. На части этих узлов и развертывается программное обеспечение системы. В рассматриваемом примере такими узлами являются клиентский компьютер и сервер запросов. Кроме того, на диаграммах развертывания могут быть показаны и другие виды узлов – элементы аппаратуры, с которым ПО лишь взаимодействует, например, *PBX* или телефонный аппарат. Этот тип диаграмм не предназначен для подробного описания аппаратной части

системы, а позволяет моделировать только ту часть оборудования, которая прямо или косвенно связана с ПО системы. В целевую систему может входить дополнительное сетевое оборудование – переключатели и т. д. Для подробной спецификации всего этого целесообразно использовать не UML, а средства классического инженерного проектирования.

Построение инженерных чертежей на сегодняшний день также компьютеризировано. Самыми распространенными программными продуктами здесь являются пакеты AutoCAD, Microsoft Visio и др.



а б

Рис. 2.4. Диаграммы развертывания: а – описательный уровень, б – экземплярный уровень

Диаграммы развертывания могут использоваться, например, как приложение к техническому заданию, а также при обсуждении цен на различные офисные АТС, телефонные аппараты и компьютеры. Такую диаграмму может составить менеджер проекта до обсуждения архитектуры системы с разработчиками. Эта диаграмма может совершенствоваться во время первых таких обсуждений. Такое «начало от аппаратуры» часто является хорошим стартом проекта, поскольку именно в терминах аппаратуры для многих программно-аппаратных систем формируется существенная часть их функциональных требований: ПО должно уметь управлять оборудованием в различных режимах и т. д. Наконец, диаграмма развертывания может давать хороший обзор всей системы, доступный для непрограммистов (особенно в составе Power-Point презентации с устными пояснениями), поскольку содержит минимум специальных деталей.

Диаграммы компонент (*component diagrams*)

При обсуждении архитектуры системы в качестве следующего промежуточного результата может появиться диаграмма, приведенная на рис. 2.5. Это – *диаграмма компонент* UML.

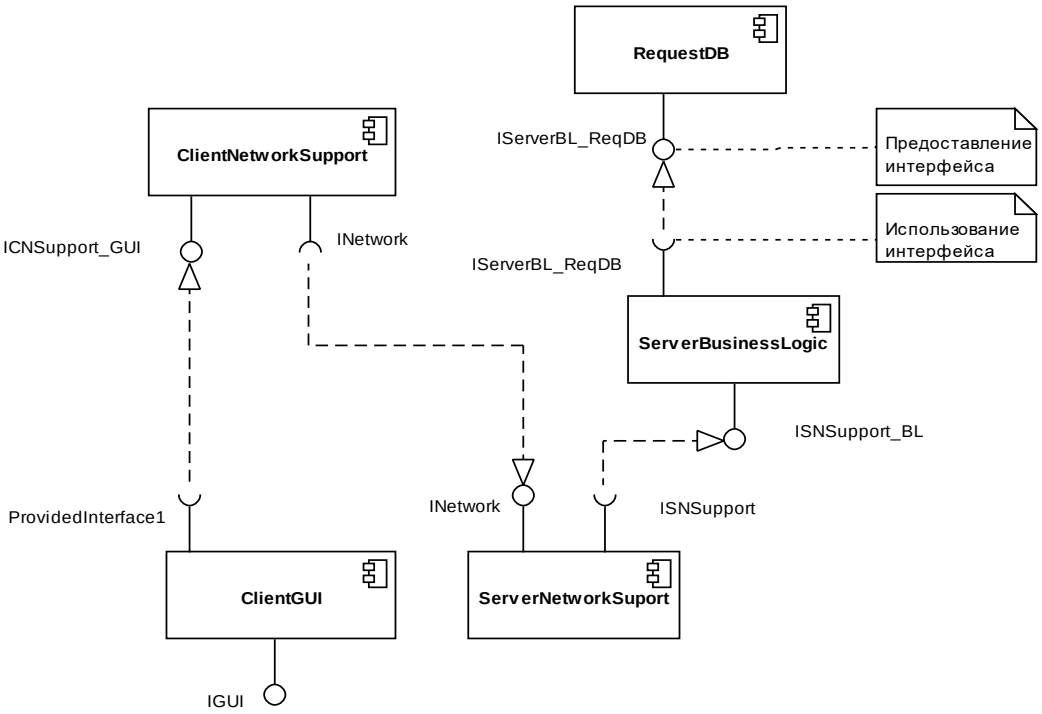


Рис. 2.5. Диаграмма компонент

На этих диаграммах представляются *компоненты* (components) – независимые модули ПО, скрывающие свою реализацию и взаимодействующие через *интерфейсы*.

Независимость компонент выражается в реализации существенно различной функциональности системы. Например, модуль ClientGUI реализует пользовательский интерфейс рабочего места оператора, модули ClientNetworkSupport и ServerNetworkSupport – поддержку сетевого взаимодействия между клиентом и сервером, модуль ServerBusinessLogic – бизнес-логику сервера, а модуль RequestDB отвечает за взаимодействие с базой данных заявок и синхронизацию с системой обработки заявок.

Каждый такой модуль независим с точки зрения физической организации – его реализация скрыта от окружения, все его взаимодействие с окружением происходит по строго определенным правилам, а сам он часто оказывается независимым бинарным файлом (например, DLL-файлом).

Возможна также независимость периода исполнения – каждая из компонент может находиться или на отдельном компьютере, или в отдельном процессе операционной системы, или работать в контексте отдельной нити (thread).

Наконец, разработку каждого такого модуля можно поручить отдельному разработчику или команде разработчиков, т. е. с помощью компонент организовать разделение коллектива программистов.

В силу своей независимости и необходимости взаимодействия компоненты имеют *интерфейсы* (interfaces), позволяющие скрыть их внутреннее устройство и предоставить определенный способ обращения к своим функциям.

Интерфейс на диаграммах UML изображается маленьким кружочком, который соединен обычной линией со своей компонентой. Использование интерфейса показывается пустой чашечкой, которая соединена обычной линией с компонентой и пунктирной линией с «потребляемым» интерфейсом.

Понятие компоненты является очень емким, и точного определения для него не существует. Неоднозначность возникает не столько в связи с разночтениями исследователей, сколько в связи с распространением различных технологий и средств программирования, использующих это понятия и по-разному его трактующих.

Самыми распространенными являются *компонентные технологии* – JavaBeans, EJB, CORBA, DCOM, .Net, web-сервисы и др. Они позволяют создавать распределенные системы, которые в связи с распространением Интернета оказываются одним из основных направлений современного программирования.

Информация, представленная на диаграмме с рис. 3.5, может со временем меняться: интерфейсы уточняются, добавляются новые компоненты, существующие разбиваются на более мелкие и т. д. Диаграммы компонент проекта целесообразно поддерживать в актуальном состоянии, (имея в виду итеративность разработки и внесение в проект всяких изменений), поскольку компонентное представление системы часто является ядром ее архитектуры. Иметь корректное и компактное описание архитектуры всегда полезно, так как с его помощью легче следить за изменениями в проекте и удерживать всю картину целиком.

Поддержку актуальности каких-либо UML-диаграм осуществить сложно. В процессе работы может появиться большое число деталей, требующих реализации. При этом возникает риск невыполненият проекта в указанные сроки, а поддержка соответствующих UML-диаграмм игнорируется. Таким образом, если невозможно ясно и кратко выразить главное в какой-либо сложной деятельности, необходимо проанализировать, где могла произойти ошибка. В данном случае имеет смысл поддерживать актуальность именно этой диаграммы, поскольку она является одной из основных спецификаций архитектуры ПО телефонной службы приема заявок.

Еще один важный аспект системы, изображенный на диаграмме – *интерфейсы* компонент. Их нужно прорабатывать особенно тщательно и вовремя, поскольку, если приложение создается разными рабочими группами, распределенными географически, то запоздалое согласование

интерфейсов может потребовать серьезных модификаций в уже написанном коде.

Диаграмма, показывающая, каким образом компоненты телефонной службы приема заявок распределяются по аппаратной части системы, представлена на рис. 2.6.

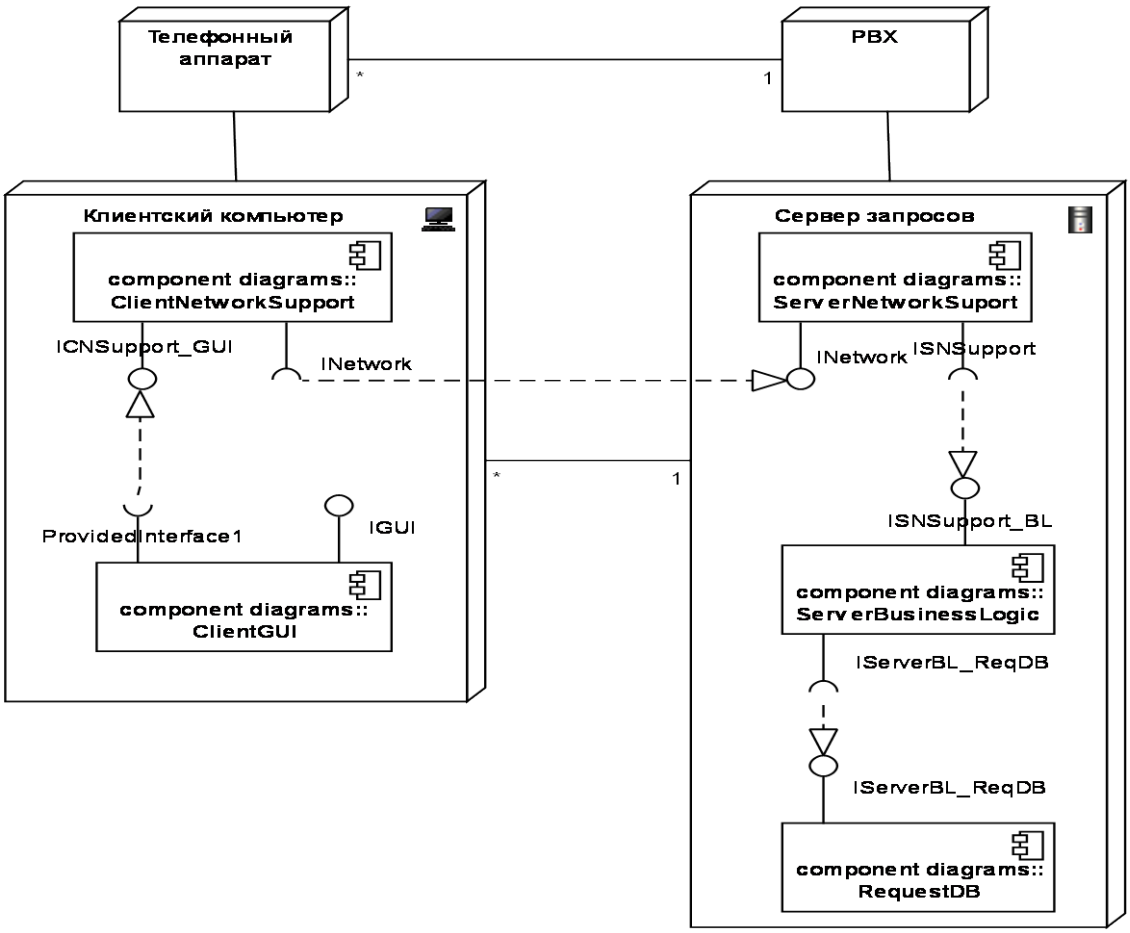


Рис. 2.6. Размещение компонент на диаграммах развертывания

Отметим, что описание типов узлов диаграмм развертывания производится на описательном, а не на экземплярном уровне.

Заметим, что именно диаграмма, изображенная на рис. 2.5, является основной в процессе разработки, так как в ней отсутствует лишняя информация. То, какие именно компоненты располагаются на сервере, а какие на клиенте – не очень важная деталь, поскольку система не очень большая. Кроме того, факт распределения компонент по аппаратуре не является здесь предметом изменений, как в более сложной системе, где существует несколько разных серверов, клиенты различных типов и т. д. Диаграмма с рис. 7.6 является, скорее, «разовой» и полезна для какого-либо отчета, для разговоров с заказчиком и т. д.

Диаграммы коммуникаций (*communication diagrams*)

В разные моменты разработки (не только при проектировании) может понадобиться прояснение определенных деталей работы системы, в особенности деталей, находящихся на стыках компонент, создаваемых отдельными членами проектной команды или рабочими группами. Побудительные причины для выяснения этих деталей различны. Например, разработчики одной из компонент могут не понять того контекста, в котором она будет работать, а специалисты по тестированию – найти ошибки, относительно которых автор каждой из компонент, задействованных в этом стыке, утверждает, что работа этих компонент корректна. Во всех этих ситуациях целесообразно собрать совещание с присутствием всех заинтересованных сторон. При этом самый заинтересованный – тестировщик, менеджер, автор компоненты, у которого возник вопрос и т. д. – готовит гипотезу того, как все должно происходить. Рекомендуется изображать эту гипотезу в виде UML-диаграммы. В данном случае используется *диаграммы коммуникаций*.

Схема поступление звонка от клиента в систему изображена на рис. 2.7. Эта диаграмма может быть полезной в случае, если нужно определить, как информация о звонке распространяется через компоненты ПО, какие процессы при этом происходят в его различных частях и какие данные передаются.

Звонок от клиента поступает на офисную АТС, затем на телефонный аппарат свободного оператора и на сервер. От сервера через локальную сеть этот звонок приходит на клиентское ПО того же оператора.

Из диаграммы становится понятно, что *PBX* должен передавать серверу вместе с информацией о звонке еще также информацию и об операторе, с которым он прокоммутировал этого клиента. Ведь сервер должен послать информацию о новом звонке на клиентское ПО именно этого оператора. Получая информацию о звонке, клиентское ПО автоматически открывает оператору специальный диалог, в который нужно ввести информацию о звонке прямо во время разговора с клиентом. Еще один важный момент, который следует из этой диаграммы: телефонный звонок на аппарате оператора должен прозвучать одновременно (или почти одновременно) с появлением на его мониторе диалогового окна для внесения информации о звонке. Все эти вопросы удобно обсуждать на фоне этой диаграммы, несмотря на то, что информация на ней представлена не в полном объеме.

На диаграммах коммуникаций изображается взаимодействие ролей классов, компонент, а не конкретные экземпляры. Роли будут подробно обсуждаться в лекции о моделировании систем реального времени. Однако отметим здесь, что роль – более общее понятие, чем объект (экземпляр), и является гнездом, куда могут быть вставлены различные объекты. В

синтаксисе UML имена ролей обозначаются без подчеркивания, а имена экземпляров – с подчеркиванием.

Диаграммы коммуникаций могут использоваться для пояснения кооперации, *композитной компоненты* или другого композитного объекта. Поэтому в ней и используются роли, а не объекты. В заголовке диаграммы указывается имя этого композитного объекта и применяется тег *comm* для обозначения диаграммы коммуникаций.

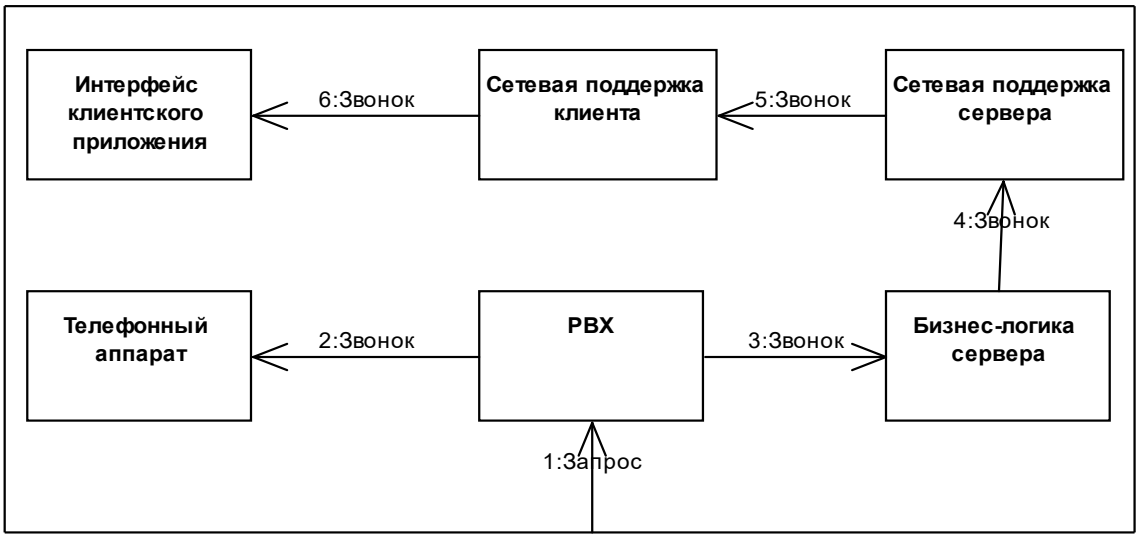


Рис. 2.7. Диаграммы коммуникаций

Диаграммы последовательностей (*sequence diagrams*)

Обратимся теперь к временным свойствам алгоритмов работы системы приема телефонных заявок. Для этого в UML есть диаграммы последовательностей.

Диаграмма, представленная на рис. 2.8, сфокусирована на действиях оператора клиентского ПО. Во-первых, на ней явно изображено, что два события – звонок оператору по телефону и появление диалога для внесения информации о звонке на дисплее оператора – должны происходить одновременно. Следовательно, необходимо тестировать это требование в условиях, идентичных условиям заказчика, – в его локальной сети, с тем быстродействием, которое она может обеспечивать, с определенным количеством одновременно работающих в сети операторов и т. д. ПО должно соревноваться по скорости с процессом коммутации в *PBX*. Вполне возможно, что звонок будет поступать намного раньше, чем соответствующая экранная форма появится на экране оператора. Это может оказаться весьма неудобным, и скорость обработки звонка сервером ПО нужно увеличивать. Кроме того, то или иное быстродействие может потребовать существенно отличающейся реализации серверных компонент, поэтому следует все продумать заранее. Создание диаграмм последовательностей помогает фиксировать переходы для алгоритмов на

этапе проектирования. Программистам нужно тщательно разрабатывать различные детали архитектуры перед началом и во время программирования, приступая к новому этапу работы. Предварительное обдумывание с фиксацией решений с помощью диаграмм и обсуждение этих диаграмм с другими сотрудниками могут предотвратить ошибки, которые, будучи допущенными, потребуют существенных усилий на исправление, намного превышающих те, что были потрачены на проектирование.

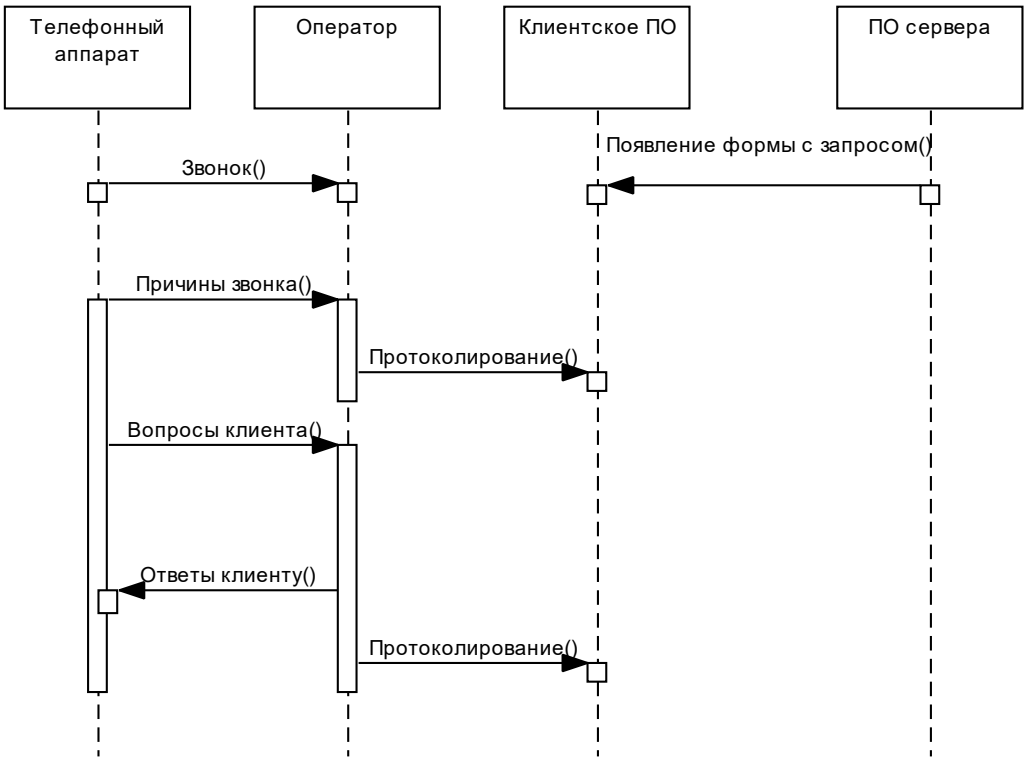


Рис. 2.8. Диаграмма последовательностей

На диаграммах последовательностей, так же как и на диаграммах коммуникаций, показываются роли классов. Фактически это одна и та же информация, но в разных видах. На диаграммах последовательностей она показана с точки зрения временного аспекта, на диаграммах коммуникаций – с точки зрения отношений взаимодействующих частей (т. е. здесь яснее выражен структурный аспект). Можно сказать, что диаграмма последовательностей является «двойником» диаграмм коммуникаций.

Временные диаграммы (*timing diagrams*)

Этот тип диаграмм является разновидностью диаграмм последовательностей и предназначен для наглядного изображения потока изменения состояний нескольких ролей (классов, компонент). Последние

являются не вертикальными, а горизонтальными, и основной упор делается на наглядное изображение их состояний, точнее, того, как они меняются во времени. Такая возможность полезна, например, при моделировании встроенных систем.

Фрагмент работы системы AccessControl, которая управляет открытием/блокированием двери в помещение по предъявлению человеком электронного ключа, показан на рис. 2.9. Видно, что система состоит из трех компонентов. Первая, *panel*, является устройством, у которого есть дисплей для отображения текущего состояния всей системы и устройство считывания электронного ключа. Исходно *panel* находится в состоянии *locked* (соответствующая надпись отображается и на дисплее). После того как человек приложил к этому устройству электронный ключ и устройство считало с него информацию, *panel* посылает ее в виде сообщения *verify* второй компоненте – *процессору* (access_processor) – и переходит в состояние *waiting*. Процессор до получения сообщения *verify* находится в состоянии *idle*, а после получения этого сообщения он переходит в состояние *verifying*. После успешного окончания проверки данных электронного ключа процессор посылает компонентам *panel* и *door* сообщения *unlocku* переходит в состояние *enable*. Компонента *panel* переходит в состояние *open.* Третья компонента, *door* (собственно, сама дверь), до этого находилась в состоянии *locked* и, получив сообщение unlock, открывается (переходит в состояние *unlock*). Открытой она остается ровно 5 секунд, после чего процессор присылает ей команду *lock* и она закрывается – снова переходит в состояние *locked*. Одновременно процессор посылает команду *lock* также и компоненте *panel*, которая переходит в свое исходное состояние *locked* и отображает слово «locked» на дисплее.

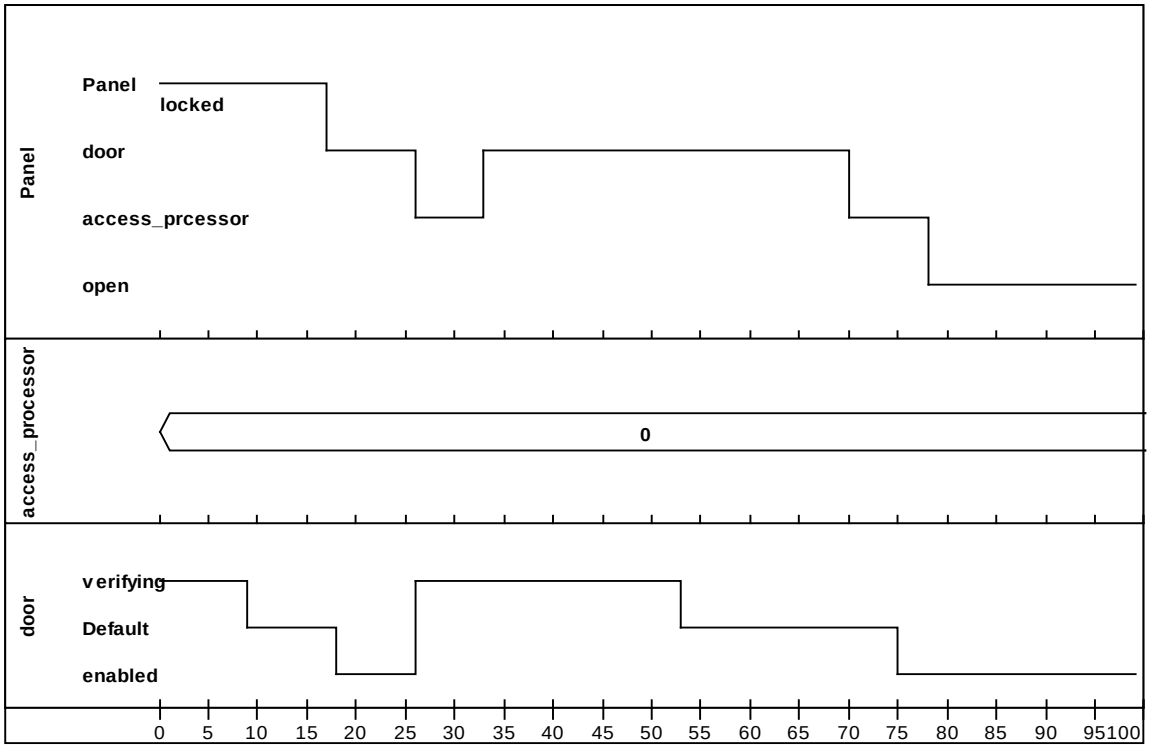


Рис. 2.9. Временная диаграмма

Видно, что на временных диаграммах, так же как на диаграммах последовательностей и коммуникаций, показываются только главные ветви алгоритмов, а ветвления отсутствуют. Компоненты и их состояния откладываются по оси ординат, время – по оси абсцисс. Время градуировано в какой-либо шкале измерений. В приведенном примере каждое деление соответствует пяти секундам.

Диаграммная область каждой компоненты – это прямоугольник, отделенный от другого, соседнего (представляющего другую компоненту), прямой линией, параллельной оси абсцисс. Компоненты могут обмениваться сообщениями, с помощью которых происходит синхронизация их поведения. Сообщения изображаются вертикальными линиями со стрелками (вверх или вниз).

Диаграммы классов (*class diagrams*)

Этот тип диаграмм является основным при разработке объектно-ориентированной системы, так как позволяет наглядно изобразить структуру классов приложения. Такие диаграммы полезны как при предварительном проектировании, так и при рефакторинге, сопровождении и исправлении ошибок, а также при изучении ПО. С их помощью легко генерировать программный код, они легко восстанавливаются по уже существующему программному коду. Кроме того, *диаграммы классов* используются при описании самих языков визуального моделирования.

Фрагмент диаграммы классов телефонной службы приема заявок представлен на рис. 2.10. Здесь показаны основные классы сервера телефонной системы обработки заявок:

CPBX_Agent – отвечает за работу с *PBX* (с аппаратурой, занимающейся коммутацией абонентов);

Coperator – содержит описание экземпляров операторов, работающих в call-центре;

CsubOperator – описывает особенного, «главного» оператора (например, начальника над группой операторов);

COperatorList – управляет множеством операторов (добавляет их в список, удаляет и т. д.);

CNetworkConnectionSupport – обеспечивает поддержку соединений сервера через локальную компьютерную сеть с компьютерами операторов;

Cdispatcher – отвечает за синхронизацию всех остальных программных сущностей на сервере.

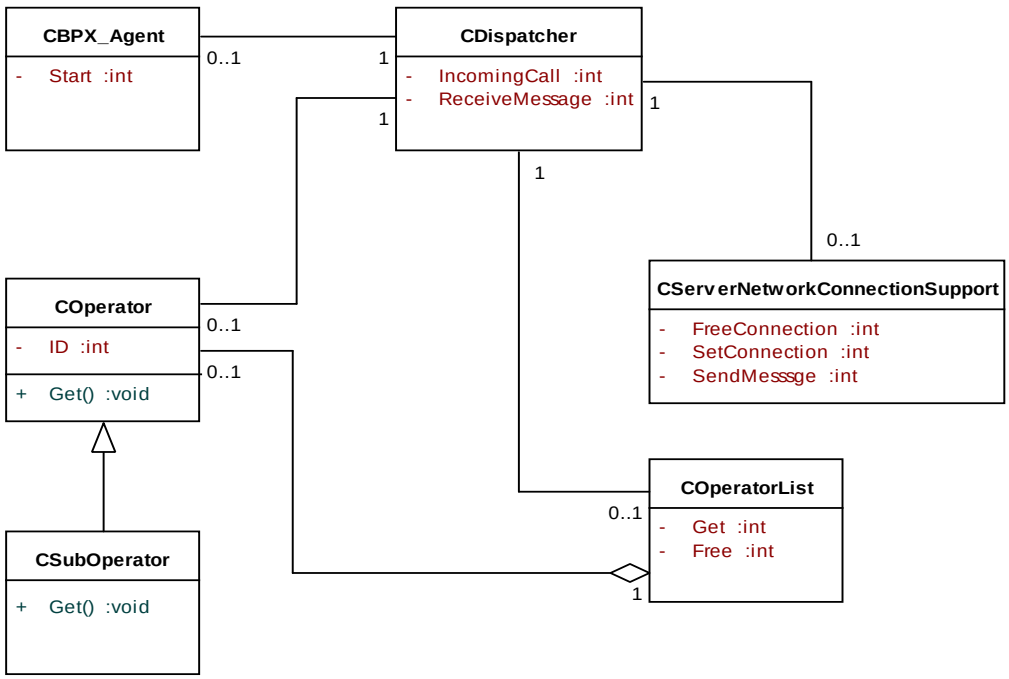


Рис. 2.10. Диаграмма классов

Итак, на диаграммах классов изображаются сами классы с атрибутами, типами атрибутов, методами, а также иерархия наследования классов. И класс, и наследование в UML полностью соответствуют конструкциям объектно-ориентированных языков программирования. Но кроме них на диаграммах классов могут также присутствовать связи между классами – *ассоциации*. Так, на рис. 2.10 Класс Cdispatcher связан с классами CPBX_Agent, COperator, CServerNetworkConnectionSupport и CoperatorList (см. рис. 2.10).

В UML существует конструкция, обобщающая класс – *классификатор* (*classifier*). С его помощью задаются элементы в модели, которые могут иметь экземпляры, операции и атрибуты. Например, экземпляры класса – это объекты.

Еще одним примером классификатора является компонента (точнее, тип компоненты). В UML-модели возможны и экземпляры компонент. Кроме класса и компоненты в UML есть и другие классификаторы.

Диаграммы пакетов (*package diagrams*)

Пакет (package) – это конструкция UML, предназначенная для упорядочения UML-моделей, а также для группировки классов.

Пакет, во-первых, выполняет служебную роль, позволяя организовать порядок в создаваемых UML-моделях, а также распределять различные модельные конструкции и диаграммы по разным «папкам».

Во-вторых, в пакеты традиционно помещают классы системы, особенно если проект большой. При этом пакеты UML могут соответствовать, например, проектам (projects) Microsoft Visual Studio. Однако пакеты UML могут быть многократно вложены друг в друга как в Enterprise Architect, а у проектов Microsoft Visual Studio такой возможности нет.

В Microsoft Visual Studio есть так называемые рабочие области (solutions), которые содержат в себе проекты. На компьютере каждого разработчика проекта могут быть созданы индивидуальные рабочие области, содержащие нужные проекты, а рабочая область всего приложения, используемая для целостной сборки приложения, может выглядеть по-другому. Таким образом, рабочие области, включающие в себя пакеты-проекты, пакетами не являются.

Пакеты связываются друг с другом специальным отношением – *зависимостью* (*dependence*). Это направленное отношение, и идет оно от зависимого пакета к независимому. Это означает, что используемый пакет содержит описание конструкций, которые зависимый пакет импортирует, а не реализует сам. Зависимость не ограничивается только *диаграммами пакетов,* но может быть использована и для связи других UML-конструкций, например, связывать два случая использования.

Примеры *диаграмм пакетов* изображены на рис. 2.11. Пакет Client содержит два пакета – ClientGUI, в котором находится описание пользовательского интерфейса, и ClientNetwork, отвечающий за сетевое взаимодействие с сервером. При этом первый пакет зависит от второго.

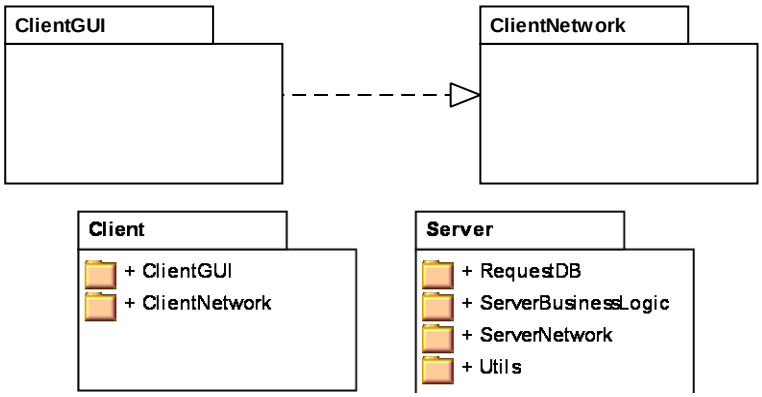


Рис. 2.11. Диаграмма пакетов

Пакет Server содержит все проекты приложения, которые реализуют работу сервера, а ServerBusinessLogic – весь код, реализующий бизнес-логику сервера. ServerNetwork реализует сетевое сообщение с клиентом, RequestDB – примитивы доступа и логику работы с базой данных запросов. Пакет Util является служебным, в нем находятся все вспомогательные типы данных, классы, операции и т. д., которые используются всеми пакетами сервера.

Содержимое пакета ServerBusinessLogic показано редствами *диаграмм классов* UML (рис. 2.12).

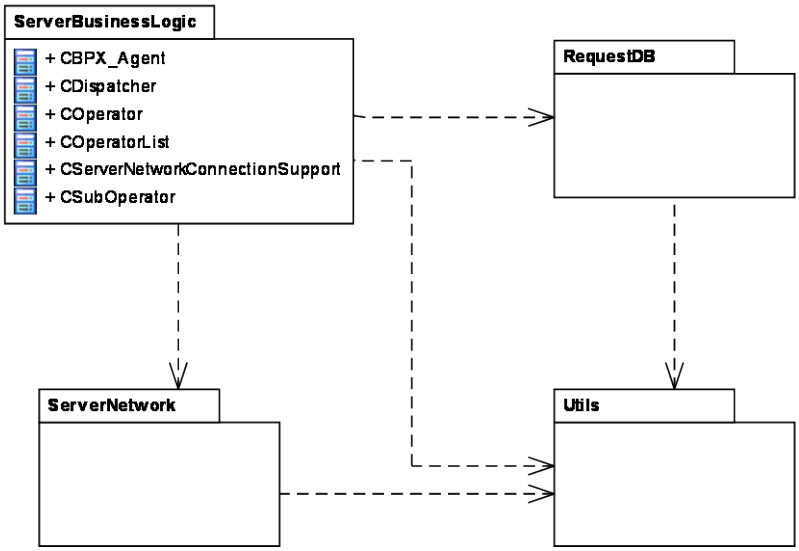


Рис. 2.12. Содержимое пакета ServerBusinessLogic в терминах диаграмм классов

В данном случае пакеты состоят из небольшого количества классов, поскольку в качестве примера представлена упрощенная модель ПО «Телефонной службы приема заявок». В действительности это приложение содержит около пятидесяти различных пакетов и около тысячи классов.

Необходимо отметить, что при проектировании приложений с большим количеством классов и пакетов целесообразно создавать *диаграммы пакетов*. Полезна предварительная оценка структуры проектов приложения. Однако ошибки при проектировании структуры пакетов большого приложения приводят к значительным неудобствам, дополнительным операциям и к потере концептуальной целостности приложения.

Диаграммы объектов (*object diagrams*)

Этот тип диаграмм предназначен для описания какого-либо фрагмента системы с помощью *объектов* (*objects*) – экземпляров классов. Объект является конкретным runtime-экземпляром некоторого класса, имеющим средство уникальной идентификации, позволяющее отличить его от других объектов того же класса, а также конкретные значения атрибутов и связей. Понятно, что все возможные экземпляры всех классов на диаграммах не изобразить, так как их слишком много. Поэтому на *диаграммы объектов* попадают только те экземпляры, которые реально существуют в каком-либо фрагменте системы в конкретный момент ее работы. Пример такой диаграммы представлен на рис. 2.13.

Здесь изображена следующая конфигурация сервера службы телефонных заявок: один диспетчер (объект: Cdispatcher»), один объект, работающий с PBX («CPBX_Agent») и два оператора – объекты «Tester1:Coperator» и «Tester2:CsubOperator». Для двух первых объектов не указаны имена, поскольку в системе одновременно может быть только по одному такому объекту. Два других объекта соответствуют тестовым операторам, один из которых является «главным» (Tester2, принадлежащий классу CSubOperator).

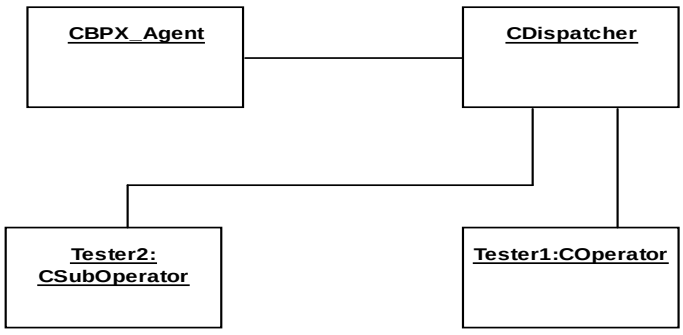


Рис. 2.13. Диаграмма объектов

Информация об экземплярах классов необходима не для спецификации системы (для этого используются, например, диаграммы классов), а для обсуждения некоторого ее фрагмента. Объект является частным случаем общей концепции экземпляров в UML. Не только классы,

но и другие сущности (например, узлы диаграмм развертывания) могут иметь экземпляры.

Общее правило для отображения имен экземпляров таково:

<Идентификатор1>: <Идентификатор2>, где <Идентификатор1> – это имя экземпляра, а <Идентификатор2> – имя его классификатора. Строка с именем должна быть подчеркнута.

У объекта может быть также секция атрибутов, где принято указывать значения для атрибутов его класса.

Объекты соединяются друг с другом *связями* (links). Так же как объекты – это экземпляры классов, так и связи – это экземпляры соответствующих ассоциаций. Однако в конкретной модели связи могут не иметь ассоциаций, например, в случае если модель объектов требуется для какого-либо документа или обсуждения и т. д. или когда строить большую модель с классами необязательно.

Кооперации (*collaborations*)

Так в UML называется описание определенной задачи (например, какой-либо пользовательской функции системы или внутренней задачи самого ПО, или же какого-либо алгоритма предметной области) в терминах взаимодействующих элементов. Описывается не поведение, а взаимодействующие стороны и их связи. Кооперации показываются на специальном типе диаграмм – на *диаграммах композитных структур* (*composite structure diagrams*).

Общающиеся стороны задаются ролями, которые описывают некоторую часть функциональности класса, используемого данным контекстом (в данном случае таким контекстом является кооперация). Например, для двух классов – Абонент и Станция – кооперация «Соединение» (рис. 2.14) определяет контекст – процедуру установки соединения между абонентом и станцией, а роли этих классов «берут» из самих классов ту функциональность, которая реализует эту процедуру. Ведь кроме этой функциональности в этих классах может быть и другая.

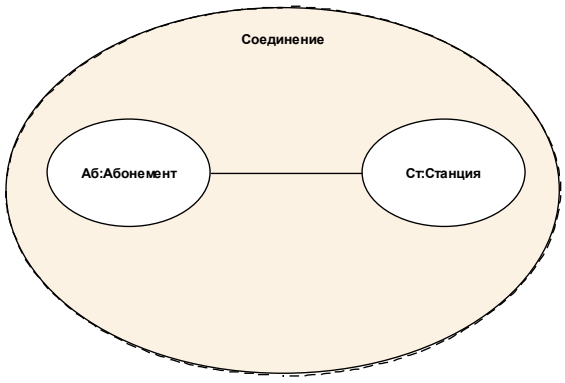


Рис. 2.14. Способ задания кооперации

Диаграммы конечных автоматов (*statechart diagrams*)

Пример *диаграммы конечных автоматов* приведен на рис. 2.15. Она описывает алгоритм поведения объектов класса Coperator системы «Телефонной службы приема заявок», изображенного на рис. 2.10.

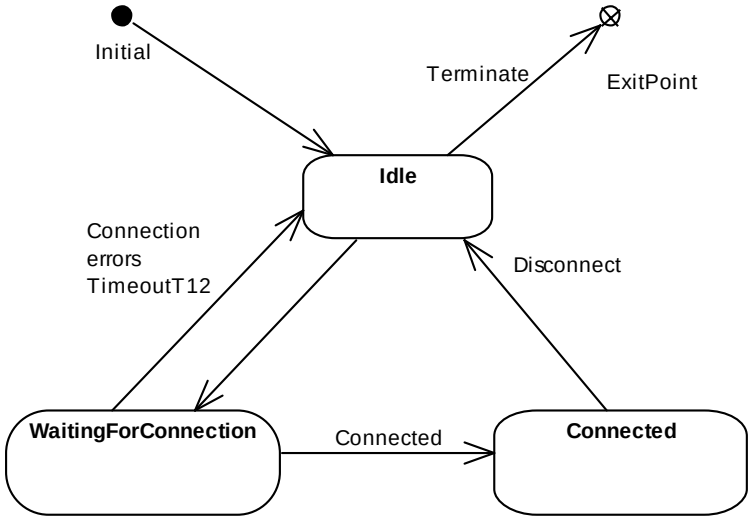


Рис. 2.15. Пример диаграммы конечных автоматов

После инициализации объекта он переходит в состояние Idle и пребывает в нем до тех пор, пока свободен и не участвует в приеме заявки от клиента. Когда приходит запрос от клиента, объект переходит в состояние WaitingForConnection – ожидание установки соединения по локальной сети с соответствующим оператором. После получения сигнала Connected объект переходит в состояние Connected, в котором пребывает на протяжении всей работы оператора с клиентом.

Из состояния WaitingForConnection объект может перейти в состояние Idle, если ожидание соединения с оператором превысит время T12.

При получении сигнала Disconnect, свидетельствующего об окончании обслуживания клиента, объект переходит в состояние Idle, в котором ожидает новый запрос. В этом же состоянии объект может обработать сигнал Terminate – указание завершить всю свою работу и освободить оперативную память.

2.2. Автоматизированная информационная система «Мониторинг деятельности застройщиков и жилищных накопительных кооперативов»

Автоматизированная информационная система (АИС) предназначена для комплексной автоматизации задач мониторинга деятельности застройщиков, привлекающих денежные средства участников долевого строительства. АИС устанавливается в организациях-застройщиках

(клиент) и в контролирующих организациях субъектов федерации (региональный узел) [65].

Автоматизации подлежат процедуры сбора и обработки регламентированной отчетности застройщиков в соответствии с Постановлением Правительства РФ от 27 октября 2005 г. № 645 о ежеквартальной отчетности застройщиков об осуществлении деятельности. Разработка технического проекта выполнена с использованием языка моделирования UML (Unified Modeling Language), признанного мировым стандартом средств моделирования и проектирования. Все диаграммы, представленные на рисунках в данной пояснительной записке, созданы с использованием программного средства Sparx Systems Enterprise Architect Version 9.2, которое гарантирует корректность разработанных диаграмм и их соответствие используемым стандартам проектирования.

Описание процесса деятельности

Состав процедур (операций) деятельности по мониторингу застройщиков описан в виде диаграмм вариантов использования (Use Case Diagramm) в нотации UML.

Классификация пользователей (актеров), участвующих в вариантах использования, приведена на рис. 2.16.

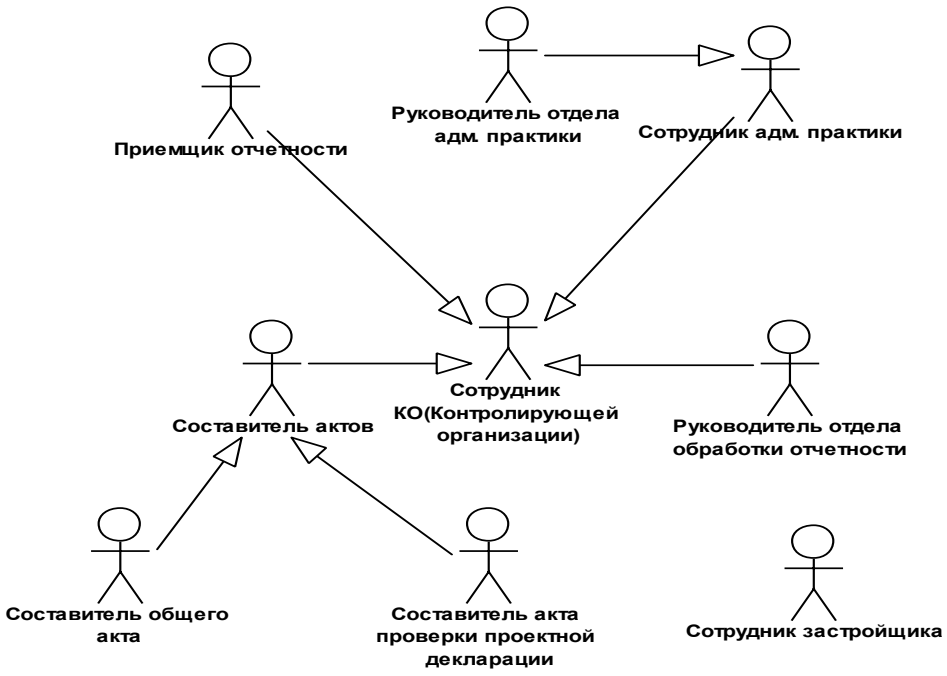


Рис. 2.16. Классификация пользователей АИС

Варианты использования по заполнению форм отчетности, предусмотренной Постановлением Правительства РФ от 27 октября 2005 г. № 645 о ежеквартальной отчетности застройщиков об осуществлении

деятельности, связанной с привлечением денежных средств участников долевого строительства, приведены на рис. 2.17.

Следует отметить, что отчетность может заполняться как сотрудником организации-застройщика, так и сотрудником контролирующей организации в случае, если отчетность застройщика была подана на традиционном (бумажном) носителе.

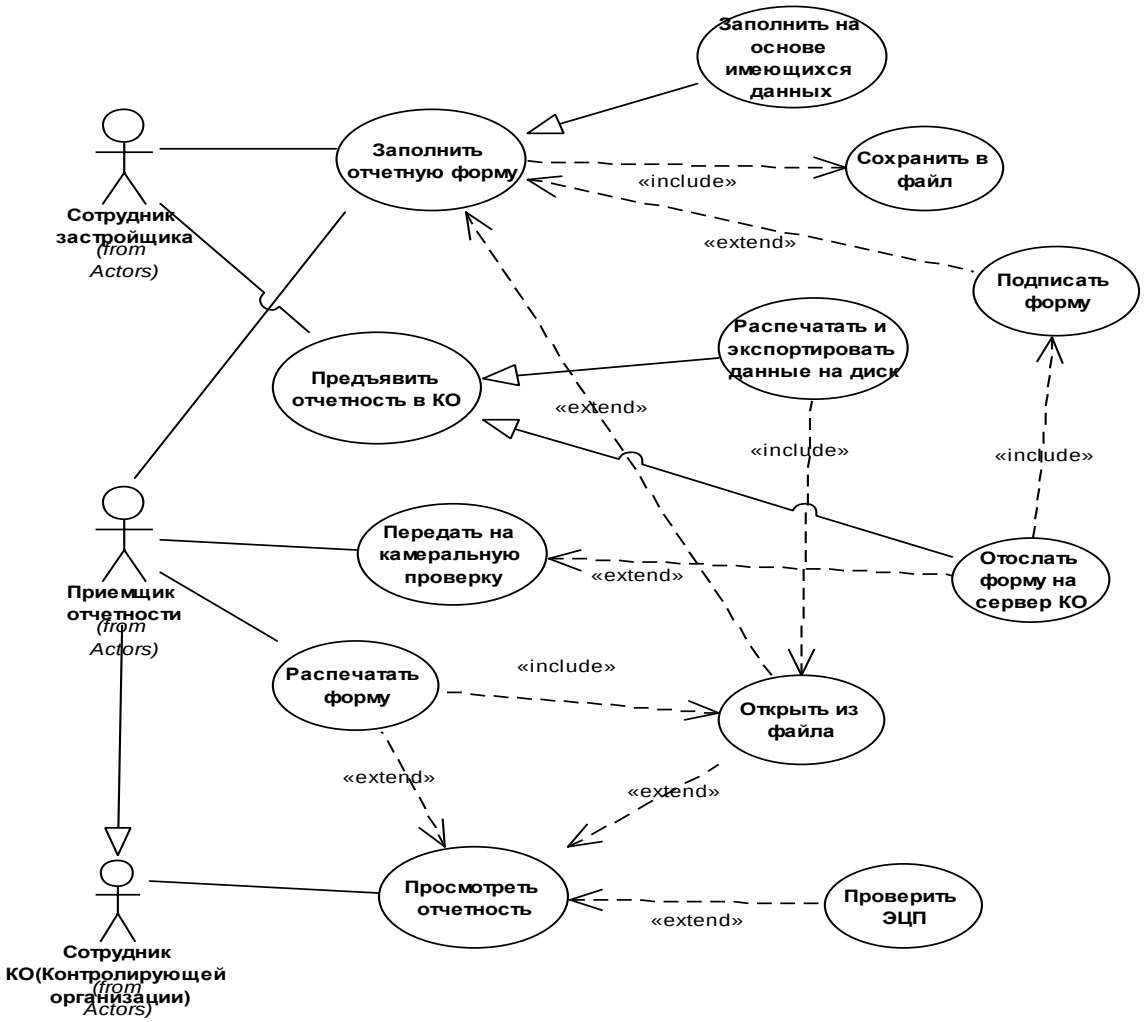


Рис. 2.17. Варианты использования по заполнению отчетности

Варианты использования, связанные с процедурами административной практики, в частности, с подготовкой документов по административной практике и учету действий по административной практике в отношении застройщиков, приведены на рис. 2.18.

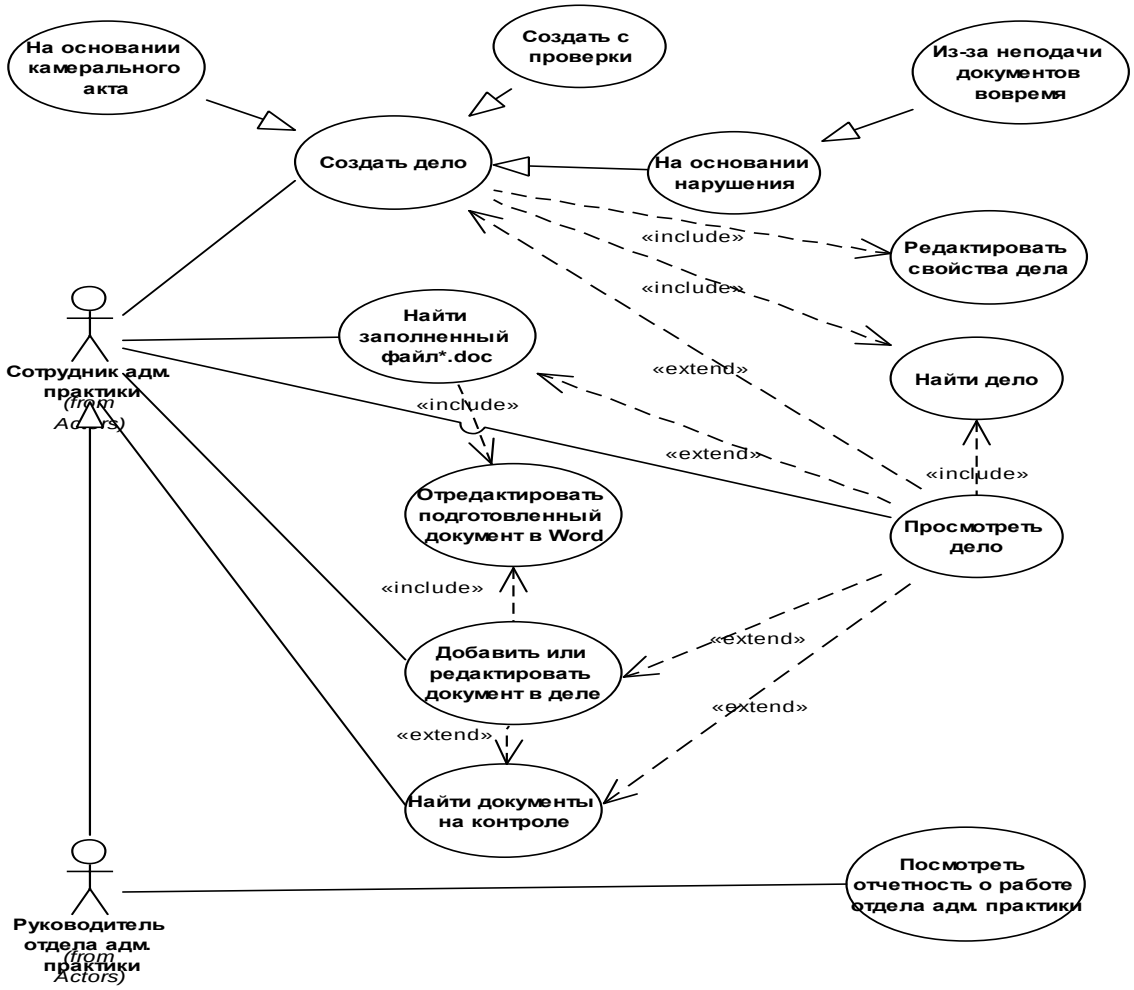


Рис. 2.18. Варианты использования по процедурам административной практики

Описание процесса деятельности ЖНК

Классификация объектов и участников деятельности

Функционирование системы связано с обработкой электронных документов. Классификация электронных документов представлена на рис. 2.19.

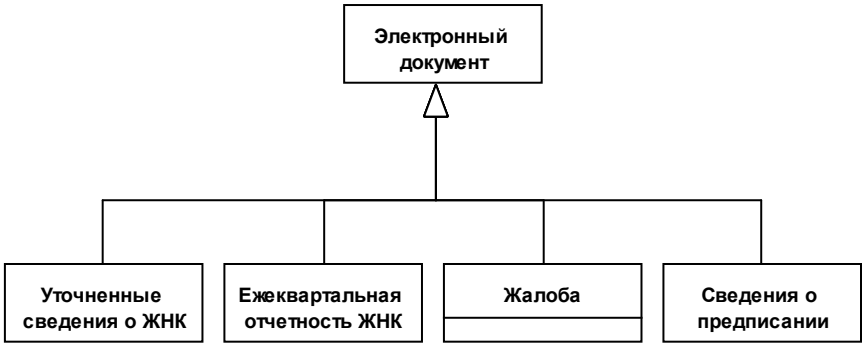


Рис. 2.19. Классификация электронных документов

В рассматриваемой системе *электронным документом* называется документ XML, соответствующий одной из известных системе схем (XSD).

Классификация участников деятельности приведена на рис. 2.20.

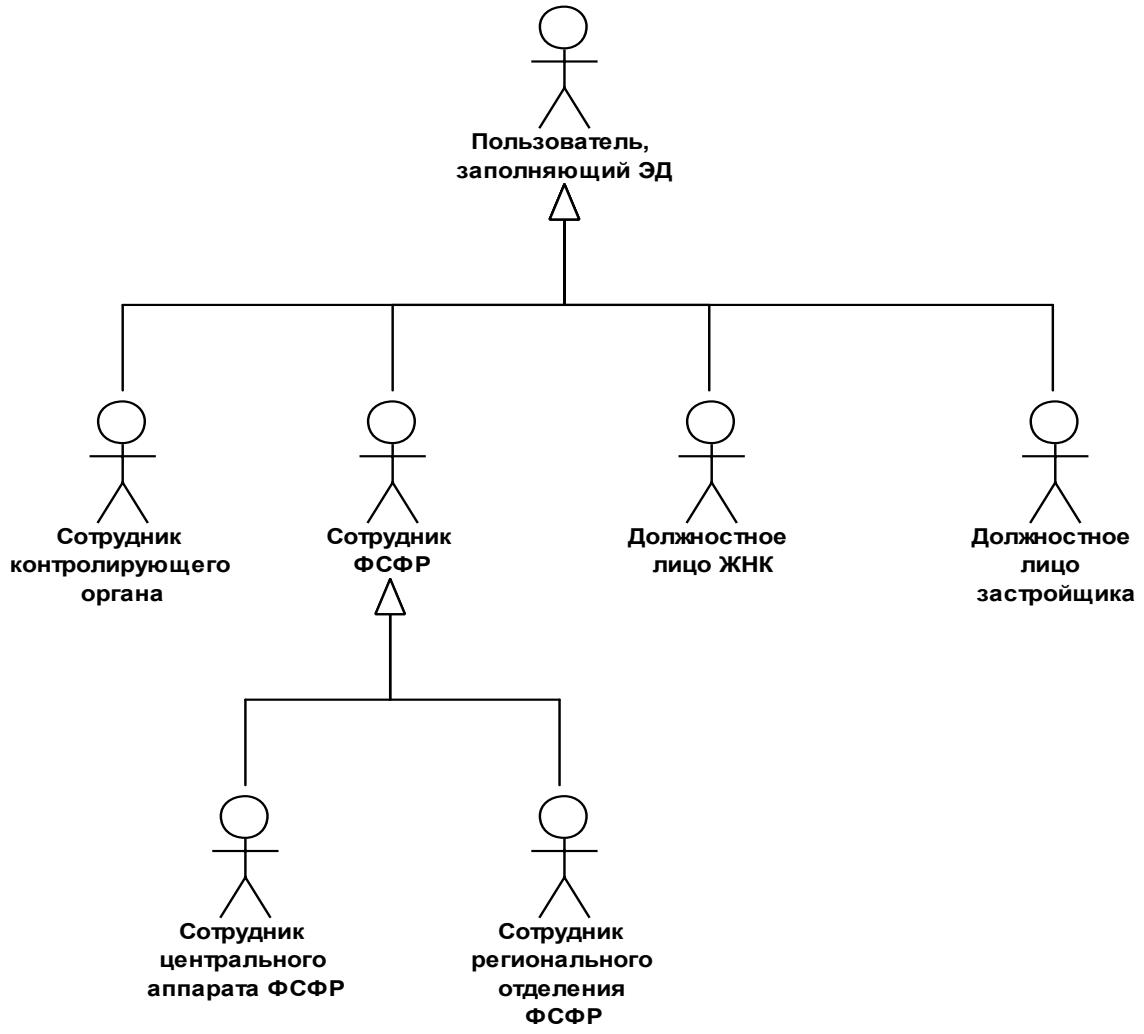


Рис. 2.20. Классификация участников деятельности

Деятельность по приему ежеквартальной отчетности ЖНК

Прием ежеквартальной отчетности приведен на рис. 2.21, а формы, необходимые для заполнения, – в табл. 2.1–2.3.

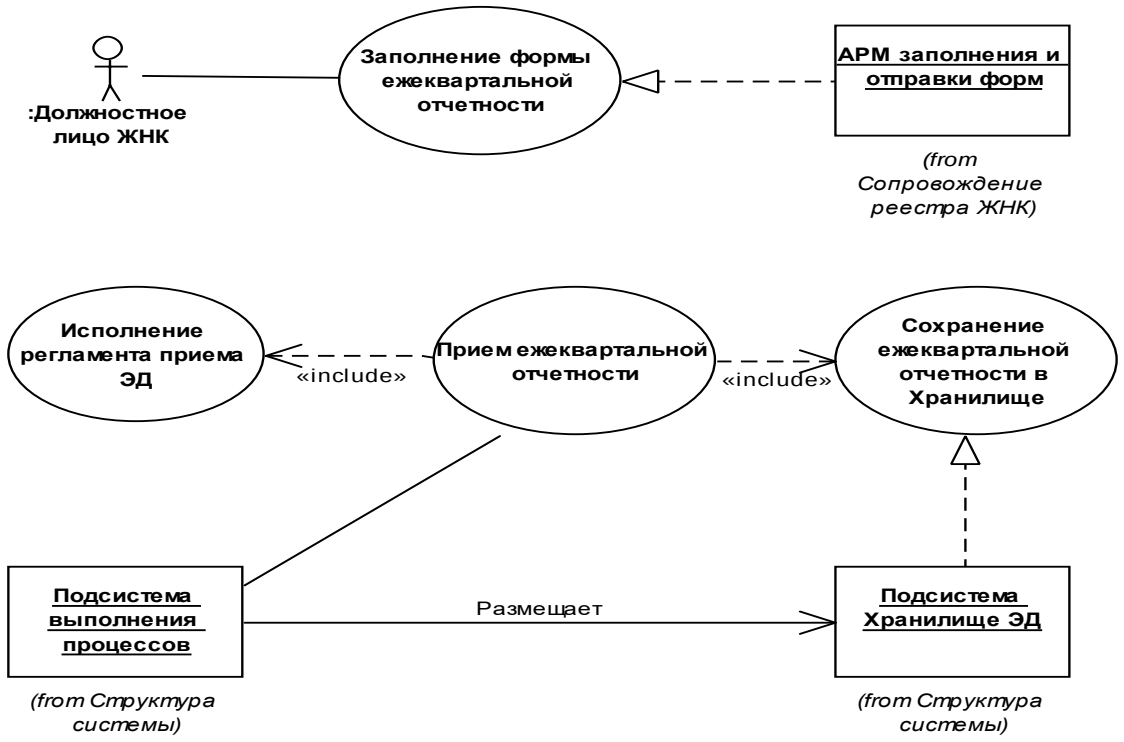


Рис. 2.21. Функциональность: прием ежеквартальной отчетности

Таблица 2.1. Заполнение формы ежеквартальной отчетности

Идентификатор	ОД_4
Наименование	Заполнение формы ежеквартальной отчетности
Пользователи	Должностное лицо ЖНК
Внешние системы	Нет
Компоненты	АРМ заполнения и отправки форм
Предусловия	Пришло время, предусмотренное Законом для подачи ежеквартальной отчетности
Постусловия	Форма заполнена и отправлена в РО
Описание сценария	Пользователь вызывает АРМ заполнения и отправки форм. Пользователь выбирает форму ежеквартальной отчетности ЖНК Пользователь заполняет поля электронного бланка. Пользователь вызывает отправку документа. Система формирует XML документ, подписывает его ЭЦП и отправляет по электронному адресу приема сообщений
Расширения	Пользователь может сохранить не до конца заполненный документ на локальный жесткий диск. Пользователь может вызвать ранее сохраненный документ и продолжить его заполнение

Таблица 2.2. Прием ежеквартальной отчетности

Идентификатор	ОД_5
Наименование	Прием ежеквартальной отчетности
Пользователи	Подсистема выполнения процессов
Внешние системы	СЭД, Подсистема Хранилище отчетности ЖНК
Компоненты	–
Предусловия	В почтовый ящик поступило сообщение от ЖНК, содержащее ЭД с ежеквартальной отчетностью ЖНК
Постусловия	Документ зарегистрирован в СЭД, ежеквартальная отчетность сохранена в хранилище
Описание сценария	Подсистема выполнения процессов обнаруживает наличие сообщения в почтовом ящике. Подсистема выполнения процессов опознает документ, как соответствующий схеме ежеквартальной отчетности ЖНК. Подсистема выполнения процессов осуществляет регистрацию документа в СЭД. Подсистема выполнения процессов инициирует выполнение деятельности «Сохранение ежеквартальной отчетности в Хранилище»

Таблица 2.3. Сохранение ежеквартальной отчетности в Хранилище

Идентификатор	ОД_6
Наименование	Сохранение ежеквартальной отчетности в Хранилище.
Пользователи	Подсистема выполнения процессов
Внешние системы	Нет
Компоненты	Подсистема Хранилище отчетности ЖНК
Предусловия	Подсистема выполнения процессов определила необходимость сохранения данных отчетности в Хранилище. Имеется документ, соответствующей схеме ежеквартальной отчетности ЖНК
Постусловия	Данные отчетности сохранены в Хранилище
Описание сценария	Подсистема выполнения процессов передает XML документ в Хранилище. Система сохраняет XML документ в виде неразобранного текста в новой строке соответствующей таблицы. Система разбирает сохраненный XML документ и сохраняет значения тэгов и атрибутов во вновь созданные строки таблиц, структурирующих документ. Система обеспечивает ссылку из таблиц, структурирующих документ, на строку таблицы, содержащую неразобранный документ

Основные технические решения

Решения по структуре системы, подсистем, средствам и способам связи для информационного обмена между компонентами системы и подсистем

Функциональная декомпозиция приведена на рис. 2.22. Структура, размещение и взаимодействие компонентов представлено на диаграмме размещения (рис. 2.23).

Состав функций, комплексов задач реализуемых системой (подсистемой)

Функции компонент, реализующих систему (2.23):

Клиент АИС МЗ – приложение .NET (Form), реализующее функции заполнения электронных бланков, преобразование данных в формат XML в соответствии со схемой (XSD), сохранение отчетности в формате XML, подписание документов ЭЦП, передачу документов по протоколу HTTP (HTTPS) в компонент приема отчетности.

Компонент приема отчетности – приложение ASP.NET, реализующее функции приема отчетности от клиента АИС МЗ по протоколу HTTP (HTTPS), сохранение отчетности в хранилище, структурирование и сохранение структурированных данных в хранилище.

Справочник застройщиков – приложение ASP.NET, реализующее операции по формированию и обслуживанию данных справочника застройщиков, включая строящиеся объекты недвижимости, и вспомогательных справочников, используемых системой.

Мастер акта камеральной проверки – приложение ASP.NET, реализующее операции проверки полученной (структурированной) очтетности, формирование документа «Акт камеральной проверки», актуализацию справочника застройщиков.

Компонент административной практики – приложение ASP.NET, реализующее функции подготовки документов административной практики на основании шаблонов и различных данных, представленных в хранилище, сохранение созданных документов в хранилище, систематизацию документов (за счет сопровождения структуры административных дел).

Менеджер аналитических форм – приложение ASP.NET, входящее в состав Microsoft Reporting Service (в свою очередь, входящего в состав Microsoft SQL Server) и реализующее функции ведения каталога аналитических отчетов и среду выполнения этих отчетов.

Хранилище данных – база данных под управлением Microsoft SQL Server, обеспечивающая хранение всех персистентных данных системы, и доступ к этим данным.

Репозитарий отчетов – база данных, хранящая описания аналитических отчетных форм.

Обозреватель Интернет – стандартный компонент операционной системы, обеспечивающий отображение данных, полученных по протоколу HTTP, и навигацию по гиперссылкам.

Офисные программы – стандартные программы, обеспечивающие отображение, редактирование и печать офисных документов.

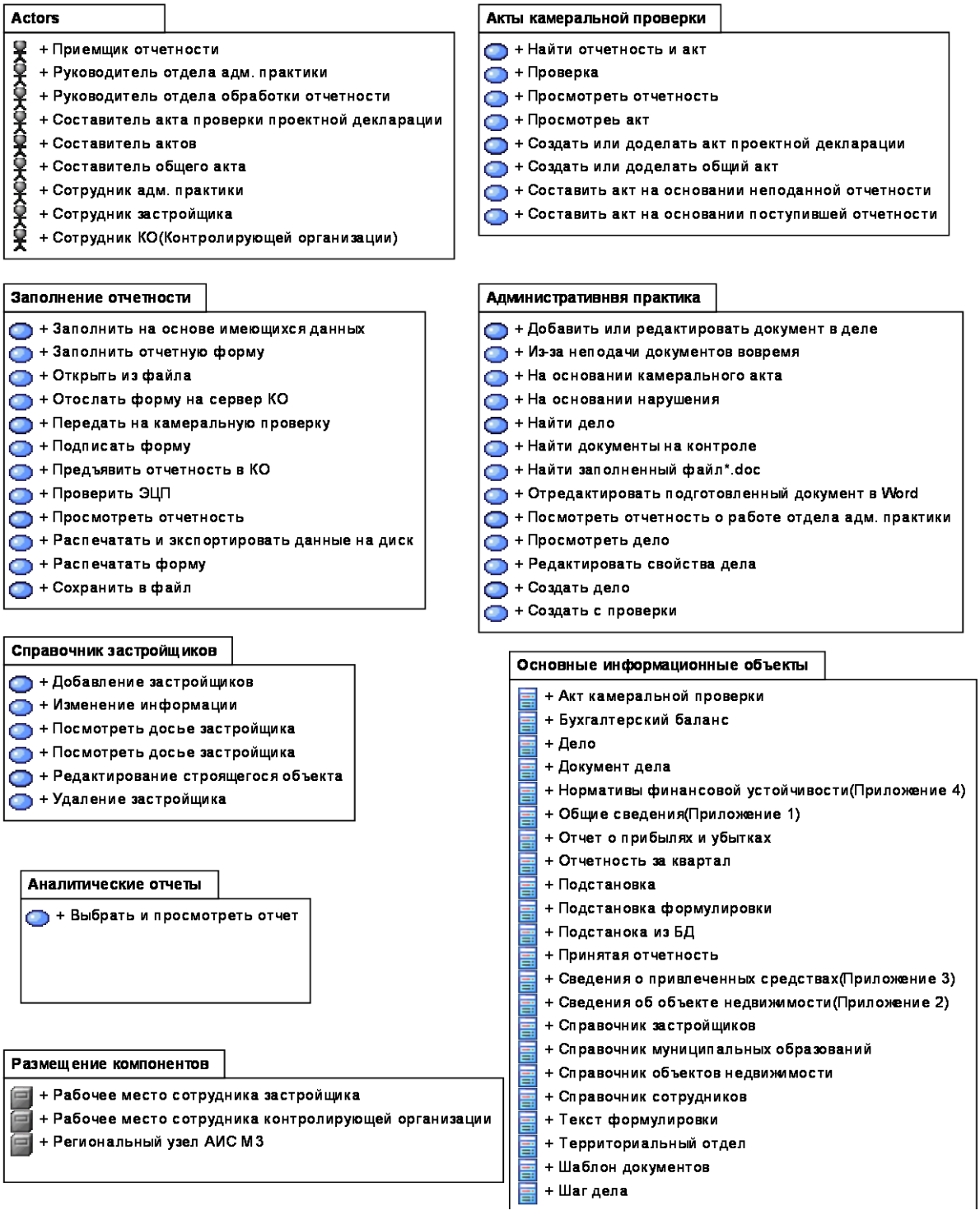


Рис. 2.22. Функциональная декомпозиция

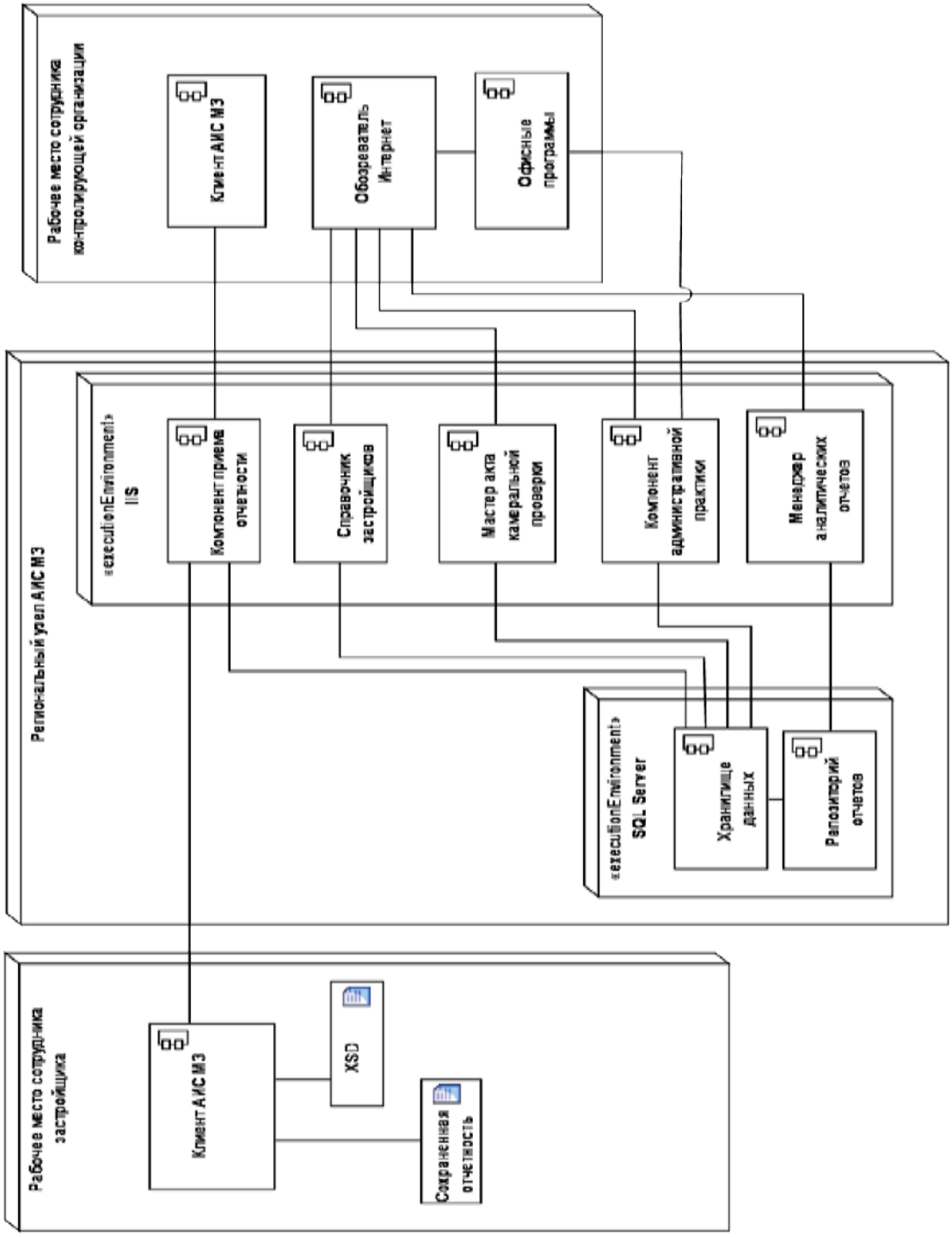


Рис. 2.23. Структура, размещение и взаимодействие компонентов

Решения по комплексу технических средств, его размещению на объекте

Программное обеспечение Клиент АИС МЗ-2 устанавливается на имеющиеся у организаций-застройщиков средства вычислительной техники, подключенные к Интернет.

Решения по составу информации, объему, способам ее организации, видам машинных носителей, входным и выходным документам и сообщениям, последовательности обработки информации и другим компонентам

Основные информационные объекты и их отношения представлены на диаграмме классов в нотации UML рис. 2.24. На этой диаграмме показаны только основные атрибуты.

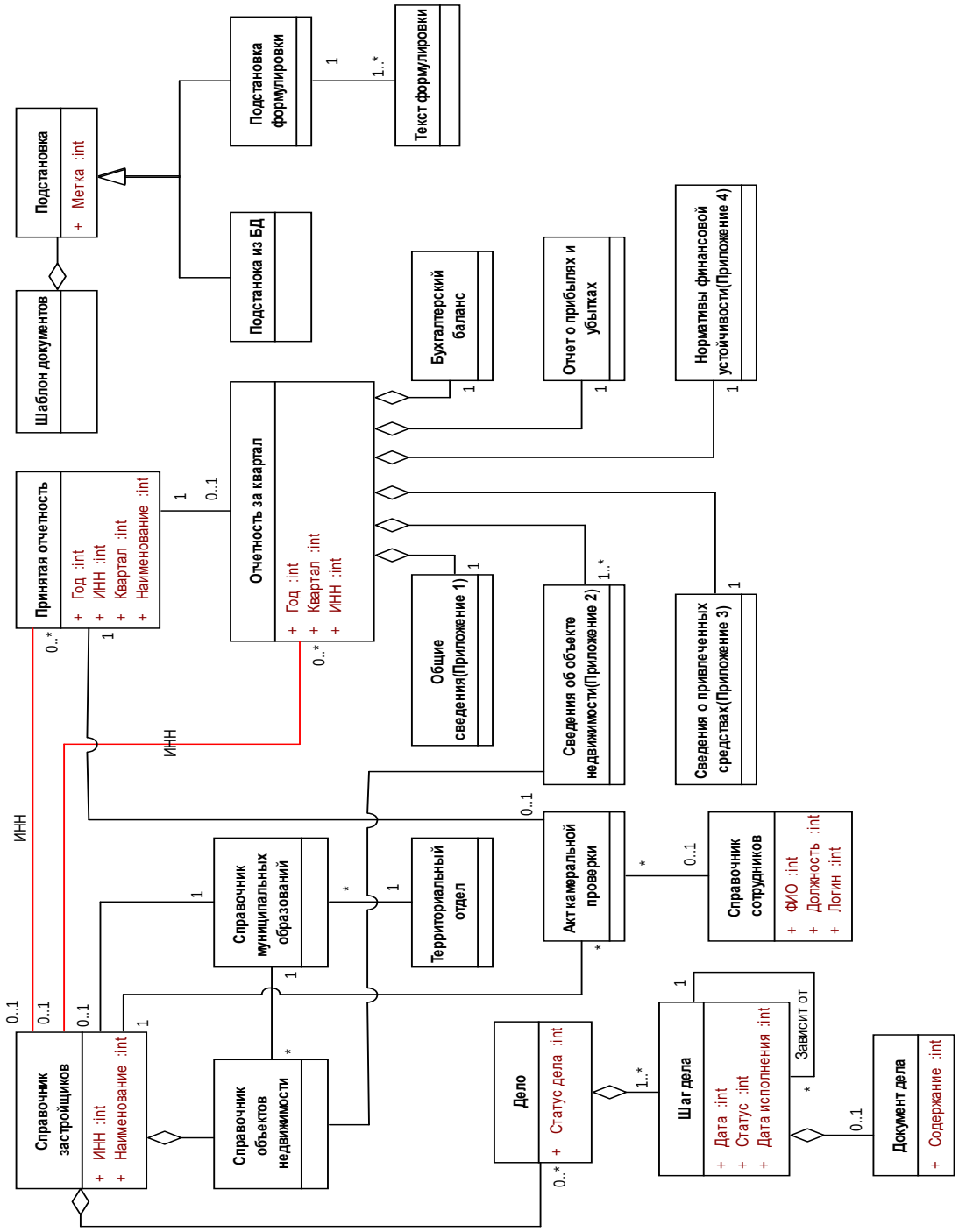


Рис. 2.24. Основные информационные объекты и их взаимосвязь (диаграмма классов)

Основные технические решения

Решения по структуре системы, подсистем, средствам и способам связи для информационного обмена между компонентами системы

Структура системы приведена на диаграмме (рис. 2.25).

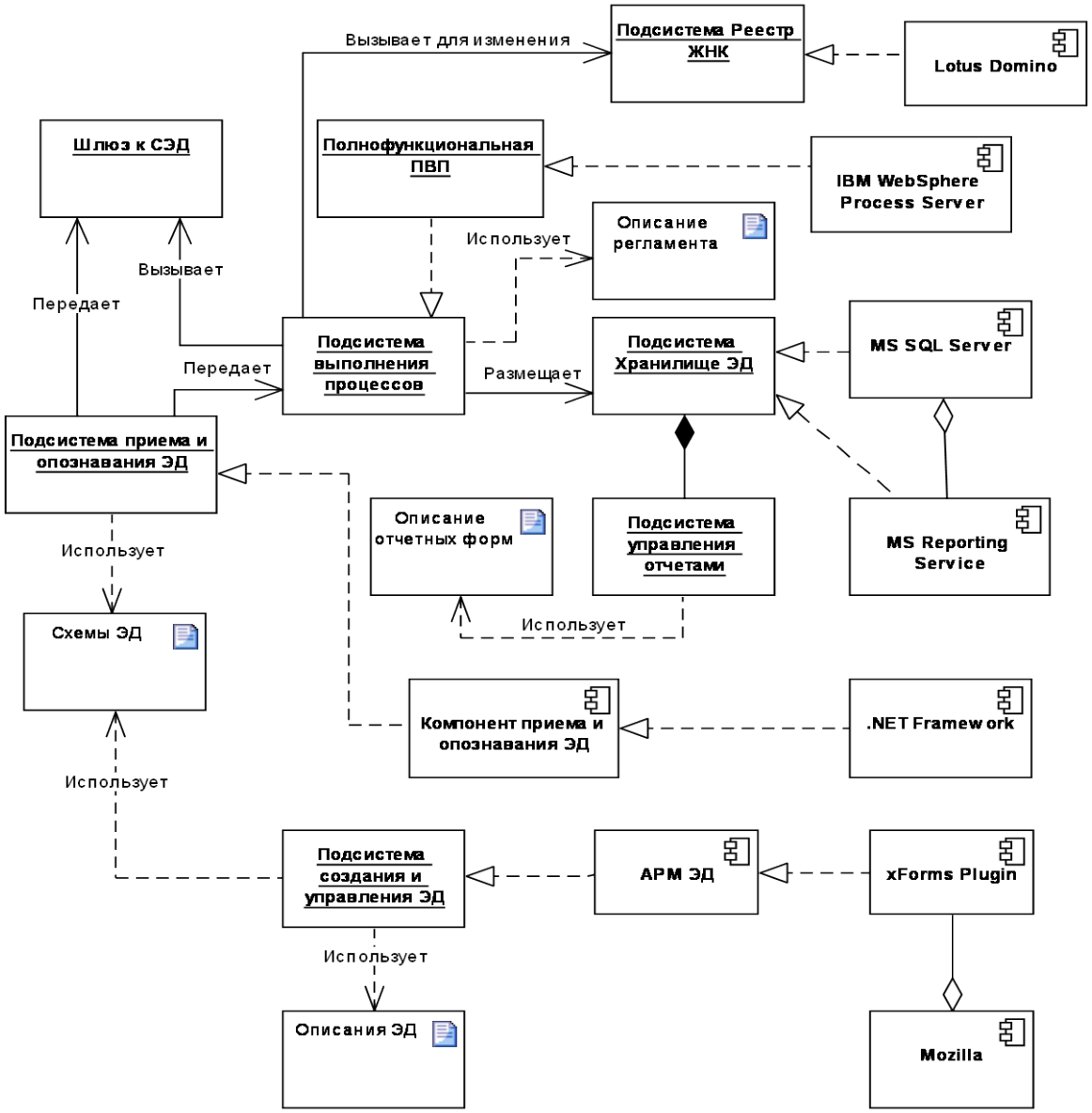


Рис. 2.25. Структура системы

Функции и назначение компонент системы

Подсистема выполнения процессов – осуществляет исполнение регламентов, представленных в виде диаграмм состояний. На вход системы поступают электронные документы, на выходе получаются управляющие воздействия на другие подсистемы.

Подсистема Реестр жилищных накопительных кооперативов (*ЖНК*) – осуществляет ведение, хранение и сопровождение реестра ЖНК на основании сведений от ФНС и данных, поступающих от ЖНК. На вход подсистемы поступают ЭД, которые могут изменять данные реестра (эти ЭД определяются подсистемой выполнения процессов). Результатом работы подсистемы является актуальный реестр ЖНК.

Шлюз к СЭД – осуществляет взаимодействие с СЭД.

Подсистема приема и опознания ЭД – отвечает за прием ЭД по электронной почте, опознание ЭД и передает опознанные ЭД на вход подсистемы выполнения процессов, а остальные в СЭД через шлюз.

Подсистема Хранилище ЭД – получает на вход ЭД и сохраняет ЭД как в исходном виде вместе с ЭЦП, так и в структурированном виде (в реляционной структуре) для последующей обработки.

Подсистема управления отчетными формами – выполняет регламентные отчетные формы по запросам пользователей либо по временному регламенту.

Подсистема создания и управления ЭД – автоматизированное рабочее место (АРМ), обеспечивающее функции создания, редактирования, подписи и отправки ЭД.

Описания отчетных форм – описания запросов к БД, форматирования и оформления результаты, а также (при необходимости) регламентов выполнения отчетных форм в формате, известном компоненту, реализующему подсистему управления отчетными формами;

Схемы ЭД – схемы XML-документов, представленные на языке XSD;

Описания ЭД – описания формата электронных форм, известные компоненту, реализующему АРМ ЭД в формате спецификаций XForms.

Решения по составу информации, способам ее организации, входным и выходным документам и сообщениям

Входные XML документы

Реестр ЖНК

Диаграмма состоит из единственного класса, экземпляр которого соответствует записи в реестре ЖНК (рис. 2.26). Назначение атрибутов приведено в табл. 2.4.

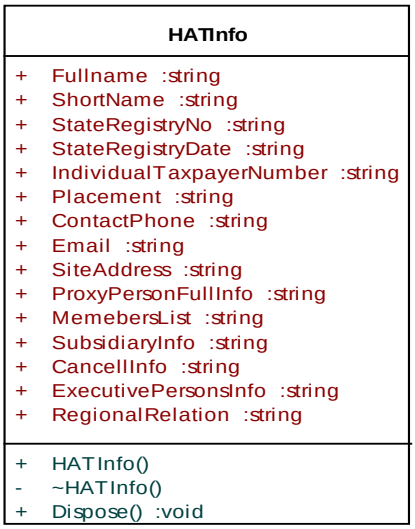


Рис. 2.26. Диаграмма классов реестра ЖНК

Таблица 2.4. Атрибуты и их назначение

Имя атрибута	Назначение
Fullname	Полное наименование юридического лица (ЖНК)
ShortName	Краткое наименование юридического лица (ЖНК)
StateRegistryNo	Код ОГРН
StateRegistryDate	Дата внесения записи в ЕГРЮЛ о государственной регистрации юридического лица при создании
IndividualTaxpayerNumber	Индивидуальный номер налогоплательщика (ИНН)
Placement	Местонахождение, а также адрес постоянно действующего исполнительного органа управления, по которому осуществляется связь с кооперативом
ContactPhone	Контактный телефон
Email	Электронная почта
SiteAddress	Адрес URL официального сайта
ProxyPersonFullInfo	ФИО, должность лица, имеющего право без доверенности действовать от имени кооператива, а также паспортные данные такого лица, ИНН (при его наличии);
MemebersList	Сведения о членах кооператива
SubsidiaryInfo	Сведения о филиалах и представительствах кооператива
CancellInfo	Сведения о прекращении деятельности кооператива, в том числе о способе прекращения его деятельности
ExecutivePersonsInfo	Сведения о лицах, являющихся исполнительным органом кооператива, членами правления кооператива и членами коллегиального исполнительного органа кооператива, а также сведения об отстранении таких лиц от занимаемых должностей с указанием причин
RegionalRelation	Принадлежность к региональному отделению (соответствующее местонахождению)

Ежекватральная отчетность ЖНК

Ежеквартальная отчетность ЖНК представлена 4 формами (в соответствии с приказом ФСФР М 06-17/пз-н от 17 февраля 2006 г). Диаграмма классов, иллюстрирующая состав ежеквартальной отчетности, изображена на рис. 2.27.

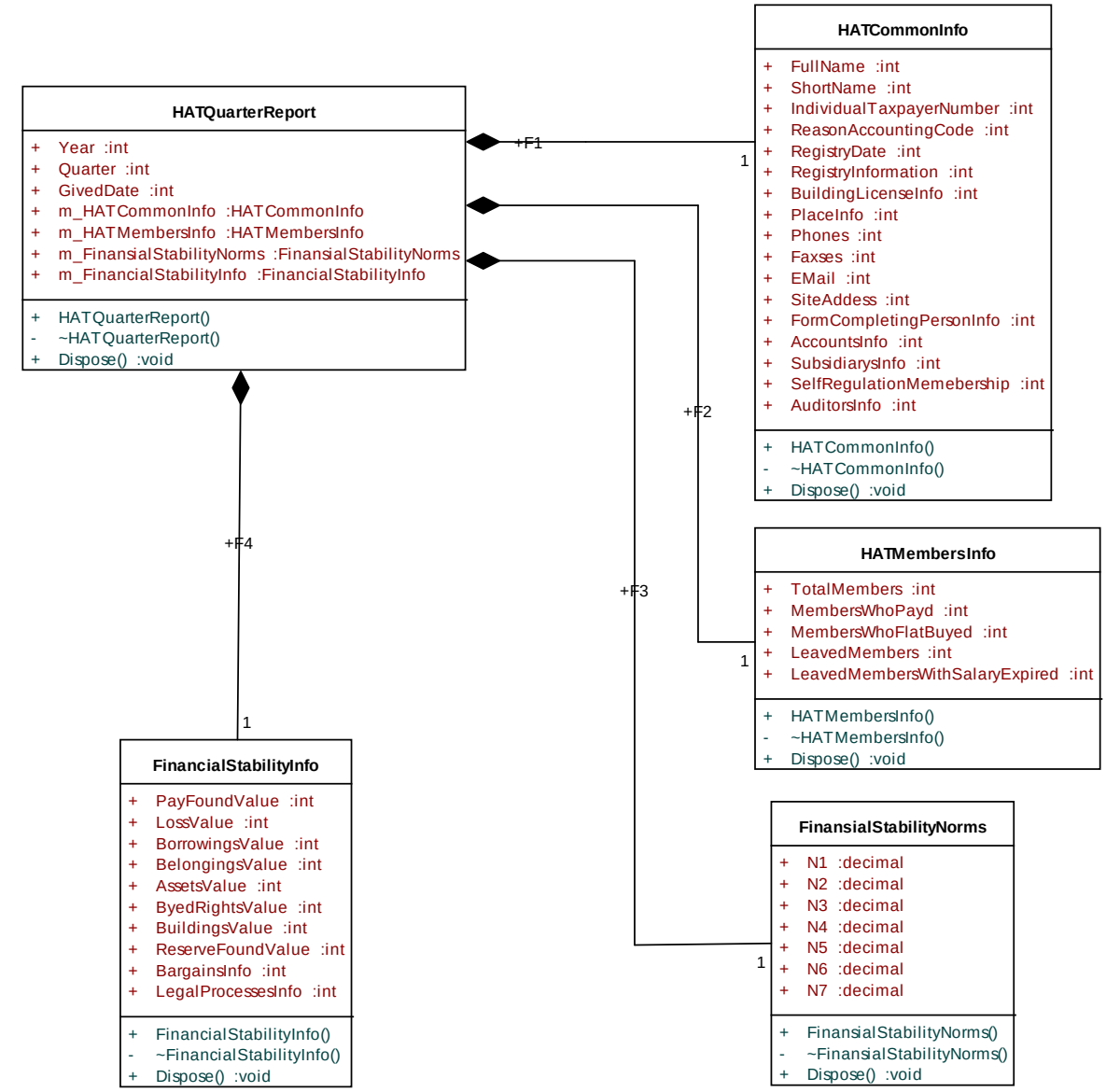


Рис. 2.27. Состав ежеквартальной отчетности ЖНК

ГЛАВА 3. Практикум «Работа с программным продуктом Enterprise Architect 9»

Практикум содержит описание 10 практических работ, выполнение которых закрепляет материал, изложенный в курсе. В нем содержатся основные сведения, необходимые для работы с одним из современных программных продуктов Enterprise Architect 9, который предназначен для создания проекта информационной системы, основанного на описании предметной области с помощью моделей различного типа. Приводится подробное описание программного продукта и обсуждается порядок проведения практических работ.

Для выполнения практических работ необходимо согласовать с преподавателем тему для индивидуальной разработки.

Аспиранту необходимо, используя теоретический материал как основу, разработать диаграммы для своей темы.

3.1. Практическая работа № 1. Общая характеристика Enterprise Architect. Рабочий интерфейс программы и операции главного меню

Цель работы: изучить основные возможности и режимы работы CASE-средства Enterprise Architect.

Рабочее пространство приложения Enterprise Architect содержит набор окон, контекстных меню и панелей инструментов. Вместе эти элементы обеспечивают простую и гибкую работу с приложением. В этой концепции рабочее пространство Enterprise Architect похоже на Microsoft Outlook и Microsoft Visual Studio.

При запуске Enterprise Architect отображается стартовая страница (рис. П.1).

Эта страница предоставляет следующие возможности:

Search – поиск объектов в Enterprise Architect. Нужно ввести имя объекта в текстовое поле и нажать кнопку […].

Getting Started – открывает Tasks Pain, отображает полезные темы и руководства для различных областей деятельности в Enterprise Architect.

Online Resources & Tutorials – открывает страницу веб-сайта компании Sparx Systems, который предоставляет доступ к широкому кругу уроков по языку UML, демонстраций, примеров, надстроек и обсуждений.

C*onfigure Options* – показывает диалоговое окно Options, которое позволяет определить, каким образом Enterprise Architect отображает и обрабатывает информацию.

Open a Project File – отображает диалоговое окно *Open Project,* которое используется для открытия существующего проекта.

Create a New Project – позволяет сохранить новый проект.

Copy a Base Project – позволяет копировать базовый проект.

Connect to Server – помогает выбрать имя источника данных для подключения.

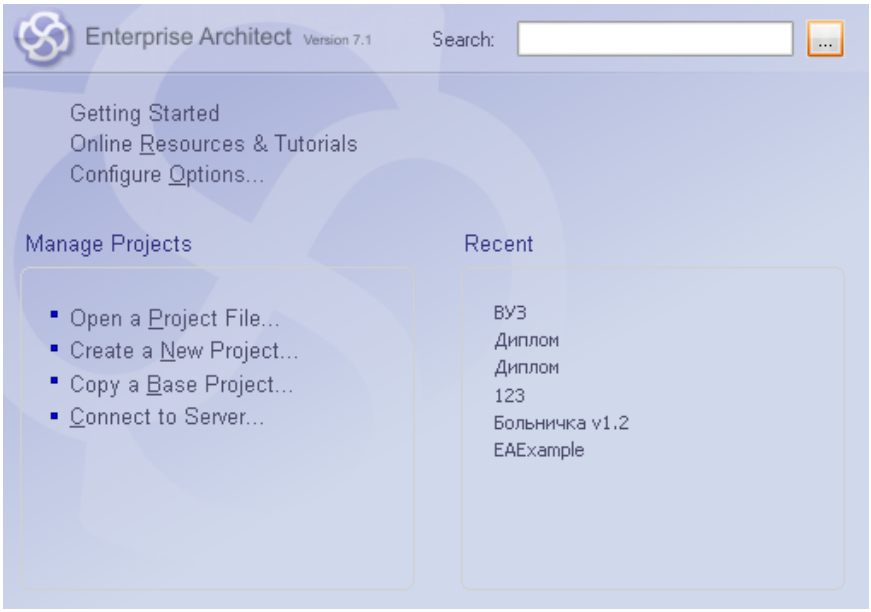


Рис. П.1. Стартовая страница Enterprise Architect

Браузер проекта (*Project Browser*) – позволяет перемещаться по пространству проекта Enterprise Architect. В нем отображаются пакеты, диаграммы, элементы и их свойства. Можно перетаскивать элементы из папки в папку или из браузера прямо на диаграмму (рис. П.2).

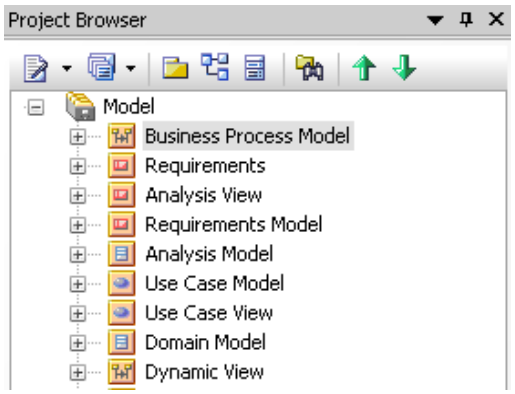


Рис. П.2. Браузер проекта

Браузер проекта служит для управления и представления всех элементов в модели. Его можно разделить на представления, каждое из которых содержит диаграммы, пакеты и другие элементы:

Представление вариантов использования включает модели вариантов использования и модели бизнес-процессов. Динамическое представление содержит диаграммы состояний, деятельности и последовательности, а логическое – модель классов.

Представление компонентов – это описание частей модели, генерирующих исполняемые DDL.

Представление размещения – физическая модель, которая показывает, какие технические средства ЭВМ использованы и какое программное обеспечение установлено.

Панель инструментов Enterprise Architect представляется как панель с пиктограммами, которые применяются для создания элементов и связей между ними. К тому же родственные элементы и связи организованы в страницы, содержащие элементы и связи, используемые в конкретном типе диаграммы (рис. П.3).

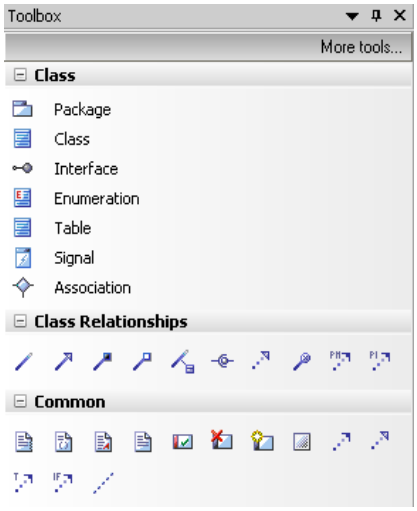


Рис. П.3. Панель инструментов для диаграммы классов

Главное меню Enterprise Architect обеспечивает управляемый мышью доступ ко многим функциям, связанным с жизненным циклом проекта наряду с функциями администрирования (рис. П.4).

Рис. П.4. Главное меню Enterprise Architect

Меню инструментов (Tools Menu) обеспечивает доступ к различным настройкам, общим для генерации кода: к управлению EAP.-файлами, правописанию, внешним ресурсам и управление ярлыками (рис. П.5).

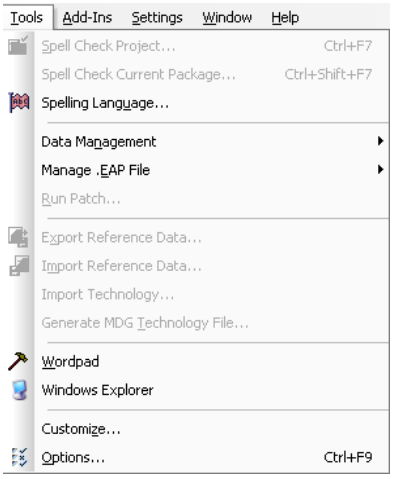


Рис. П.5. Меню инструментов (Tools Menu)

В меню инструментов наибольший интерес представляет пункт Options. Здесь находятся опции настройки Enterprise Architect для отображения и работы с моделями и элементами модели. Основные режимы представлены в табл. П.1.

Таблица П.1. Назначение вкладок пункта меню Options

Вкладка	Назначение
General	Общие настройки проекта, например, автор, адрес домашнего веб-сайта, общие настройки браузера проекта
Standard Colors	Позволяет установить цвет ряда объектов и их фон
New Diagram Defaults	Позволяет конфигурировать опции для новых диаграмм и общего поведения диаграммы
Diagram Appearance	Позволяет определить как диаграммы и их содержимое отображается на экране
Diagram Behavior	Позволяет определить, как диаграмма реагирует на действия, производимые над ней
Diagram Sequence	Позволяет установить настройки шрифта и фокус контрольного индикатора для диаграммы последовательности
Objects	Позволяет установить, как элементы выглядят на диаграмме
Links	Настройки создания, поведения и нотации связей
Communication Colors	Позволяет установить цвета, используемые в диаграммах взаимодействия
XML Specifications	Позволяет установить настройки для работы с XML
Source Code Engineering	Описывает общие настройки, применяемые для всех языков при генерации кода из Enterprise Architect
C++ (рис. П.6)	Установка опций для генерации кода на C++ включает: установку опций для текущей модели; установку опций для текущего пользователя

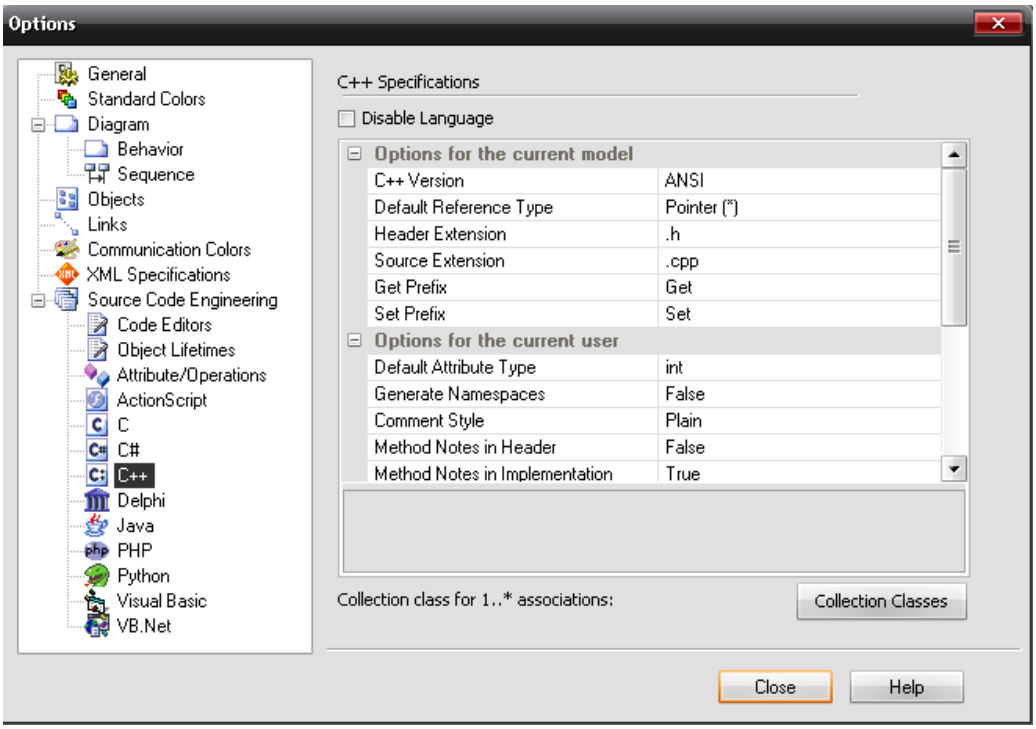


Рис. П.6. Вкладка C++ Specifications

3.2 Практическая работа № 2. Анализ предметной области

Цель работы: описать основные требования к АСОИ и сформировать глоссарий проекта.

Анализ предметной области «Система регистрации для вуза»

Перед руководителем информационной службы университета ставится задача; разработать новую клиент-серверную систему регистрации студентов, которая должна позволять студентам записываться на курсы и просматривать свои табели успеваемости с персональных компьютеров, подключенных к локальной сети университета.

Из-за недостатка средств университет не в состоянии заменить сразу всю существующую систему. Остается функционировать в прежнем виде база данных, содержащая всю информацию о курсах (каталог курсов), которая поддерживается реляционной СУБД. Новая система будет работать с существующей БД в режиме доступа без обновления.

В начале каждого семестра студенты могут запросить каталог курсов, предлагаемых в семестре. Информация о каждом курсе должна включать имя профессора, наименование кафедры и требования к предварительному уровню подготовки.

Новая система должна позволять студентам выбирать 4 курса в предстоящем семестре. Дополнительно каждый студент может указать 2 альтернативных курса на тот случай, если какой-либо из выбранных окажется уже заполненным или отмененным. На каждый курс может

записаться не более 10 и не менее 3 студентов (если менее 3, то курс будет отменен). В каждом семестре существует период, когда студенты могут изменить свои планы, поэтому нужно обеспечить им доступ к системе, чтобы добавить или удалить выбранные курсы. После того как процесс записи некоторого студента завершен, система регистрации направляет информацию в расчетную систему, чтобы студент мог внести плату за семестр. Если курс окажется заполненным в процессе регистрации, студент должен быть извещен об этом до окончательного формирования его личного учебного плана.

В конце семестра студентам нужно просмотреть электронные табели успеваемости. Поскольку эта информация конфиденциальна, система должна обеспечить защиту от несанкционированного доступа.

Профессора необходим доступ к онлайновой системе, чтобы указать курсы, которые они будут читать, просмотреть список записавшихся студентов, а также проставить оценки.

Составление глоссария проекта

Глоссарий предназначен для описания терминологии предметной области. Он может быть использован как неформальный словарь данных системы (табл. П.2).

Таблица П.2. Глоссарий проекта

Термин предметной области	Описание
Курс	Учебная дисциплина, предлагаемая университетом
Конкретный курс (Course Offering)	Определенный курс, который читается в конкретном семестре (один и тот же курс может вестись в нескольких параллельных сессиях) и включает точные дни недели и время
Каталог курсов	Полный перечень всех курсов, предлагаемых университетом
Расчетная система	Система обработки информации об оплате за курсы
Оценка	Результат, полученный студентом за конкретный курс
Профессор	Преподаватель университета
Табель успеваемости (Report Card)	Все оценки за все курсы, полученные студентом в данном семестре
Список курса (Roster)	Список всех студентов, записавшихся на конкретный курс
Студент	Личность, проходящая обучение в университете
Учебный график (Schedule)	Курсы, выбранные студентом в текущем семестре

Описание дополнительных спецификаций

Назначение дополнительных спецификаций – определить требования к системе регистрации курсов, которые не отражены в модели вариантов использования. Вместе они образуют полный набор требований к системе.

Дополнительные спецификации определяют нефункциональные требования к системе, такие как надежность, удобство использования, производительность, сопровождаемость, а также ряд функциональных требований, являющихся общими для нескольких вариантов использования.

Функциональные возможности: система должна обеспечивать многопользовательский режим работы. Если конкретный курс оказывается заполненным в то время, когда студент формирует свой учебный график с учетом этого курса, то система должна известить его об этом.

Удобство использования: пользовательский интерфейс должен быть совместимым с Windows XP.

Надежность: система должна быть всегда в работоспособном состоянии, время простоя – не более 10 %.

Производительность: система должна поддерживать до 2000 одновременно работающих с центральной базой данных пользователей, и до 500 пользователей, одновременно работающих с локальными серверами.

Безопасность: система должна запрещать студентам изменение чужих учебных графиков, а профессорам – модифицировать конкретные курсы, выбранные другими профессорами. Только профессора имеют право ставить студентам оценки, а изменять любую информацию о студентах может только регистратор.

Проектные ограничения: система должна быть интегрирована с существующей системой каталога курсов, функционирующей на основе реляционной СУБД.

Создание модели вариантов использования

Действующие лица:

Student (Студент) – записывается на курсы;

Professor (Профессор) – выбирает курсы для преподавания;

Registrar (Регистратор) – формирует учебный план и каталог курсов, ведет все данные о курсах, профессорах и студентах;

Billing System (Расчетная система) – получает от данной системы информацию по оплате за курсы;

Course Catalog (Каталог курсов) – передает в систему информацию из каталога курсов, предлагаемых университетом.

3.3 Практическая работа № 3. Разработка диаграммы вариантов использования и редактирования свойств ее элементов

Цель работы: изучить технологии формирования диаграммы вариантов использования.

Создание действующих лиц в среде Enterprise Architect

Чтобы поместить действующее лицо в браузер, щелкнуть левой кнопкой мыши на панели инструментов Toolbox по кнопке Actor, затем – по рабочей области диаграммы, в появившемся окне ввести имя актера, выбрать стереотип. В браузере появится новое действующее лицо. Слева от его имени можно увидеть пиктограмму действующего лица UML. После создания действующих лиц сохранить модель под именем coursereg (analysis) с помощью пункта меню File > Save Project As.

Исходя из потребностей действующих лиц, выделяются следующие варианты использования:

Login (Войти в систему);

Register for Courses (Зарегистрироваться на курсы);

View Report Card (Просмотреть табель успеваемости);

Select Courses to Teach (Выбрать курсы для преподавания);

Submit Grades (Проставить оценки);

Maintain Professor Information (Ввести информацию о профессорах);

Maintain Student Information (Ввести информацию о студентах);

Close Registration (Закрыть регистрацию).

Создание вариантов использования в среде Enterprise Architect

Поместить вариант использования в браузер. Для этого щелкнуть левой кнопкой мыши на панели инструментов Toolbox по кнопке Use Case. затем по рабочей области диаграммы, в появившемся окне ввести название варианта использования, выбрать его стереотип. В браузере появится новый вариант использования. Слева от него будет видна пиктограмма варианта использования UML.

Диаграмма вариантов использования

Создать диаграмму вариантов использования для системы регистрации. Требуемые для этого действия подробно перечислены далее. Готовая диаграмма вариантов использования должна выглядеть как на рис. П.7.

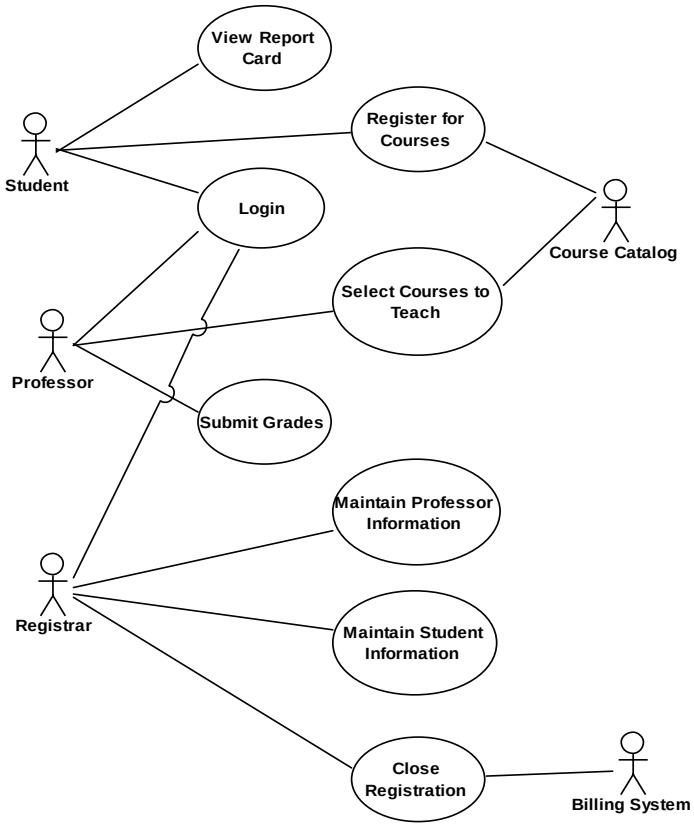


Рис. П.7. Диаграмма вариантов использования для системы регистрации

В среде Enterprise Architect диаграммы вариантов использования создаются в представлении вариантов использования. Главная диаграмма предлагается по умолчанию. Для моделирования системы можно затем разработать столько дополнительных диаграмм, сколько необходимо.

Чтобы получить доступ к главной диаграмме вариантов использования, рядом с представлением вариантов использования в браузере щелкнуть на значке «+». Это приведет к открытию представления. Открыть главную диаграмму, дважды щелкнув на ней.

Для создания новой диаграммы вариантов использования щелкнуть правой кнопкой мыши на пакете представления вариантов использования в браузере, из всплывающего меню выбрать пункт Add > Add Diagram. Выделив новую диаграмму, ввести ее имя, затем открыть ее, дважды щелкнув на названии диаграммы в браузере.

Построение диаграммы вариантов использования

Открыть диаграмму вариантов использования. Для размещения действующего лица или варианта использования на диаграмме перетащить его мышью из браузера на диаграмму вариантов использования.

Наличие общего варианта использования Login для трех действующих лиц позволяет обобщить их поведение и ввести новое действующее лицо Any User. Модифицированная диаграмма вариантов использования показана на рис. П.8.

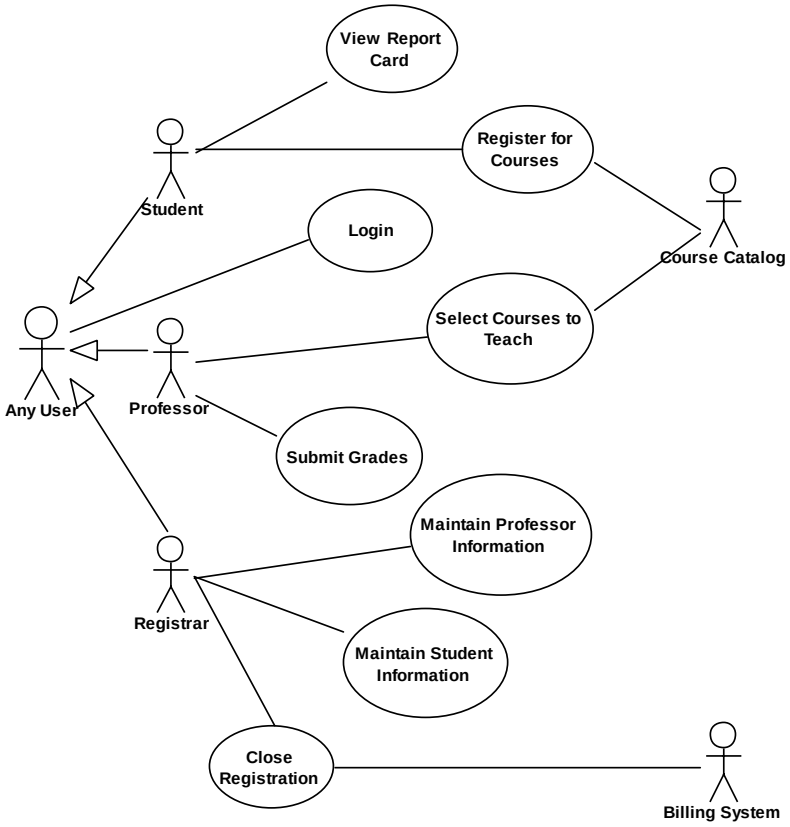


Рис. П.8. Модифицированная диаграмма вариантов использования

Добавление описаний к вариантам использования

Щелкнуть правой кнопкой мыши по варианту использования «Register for Courses». Выбрать Add > Note, ввести к этому варианту использования описание: «This use case allows a student to register for courses in the current semester» (Этот вариант использования дает студенту возможность зарегистрироваться на курсы в текущем семестре).

Описание основных и альтернативных потоков событий

С помощью MS Word создать три текстовых файла с описаниями вариантов использования Login (Войти в систему), Register for Courses (Зарегистрироваться на курсы) и Close Registration (Закрыть регистрацию).

Вариант использования Login.

Краткое описание – описывает вход пользователя в систему регистрации курсов.

Основной поток события – начинает выполняться, когда пользователь хочет войти в систему регистрации курсов. Система запрашивает имя пользователя и пароль, проверяет их, а затем открывает доступ в систему.

Альтернативные потоки – неправильное имя/пароль. Если во время выполнения основного потока обнаружится, что пользователь ввел неправильное имя и/или пароль, система выводит сообщение об ошибке. Пользователь может вернуться к началу основного потока или отказаться

от входа в систему, при этом выполнение варианта использования завершается.

Предусловия. Отсутствуют.

Постусловия. Если вариант использования выполнен успешно, пользователь входит в систему. В противном случае состояние системы не изменяется.

Вариант использования Register for Courses.

Краткое описание. Позволяет студенту зарегистрироваться на конкретные курсы в текущем семестре. Студент может изменить свой выбор (обновить или удалить курсы), если изменение выполняется в установленное время в начале семестра. Система каталога курсов предоставляет список всех конкретных курсов текущего семестра.

Основной поток событий. Данный вариант использования начинает выполняться, когда студент хочет зарегистрироваться на конкретные курсы или изменить свой график курсов.

Система запрашивает требуемое действие (создать график, обновить график, удалить график). Когда студент указывает действие, выполняется один из подчиненных потоков (создать, обновить, удалить или принять график).

Создать график. Система выполняет поиск доступных конкретных курсов в каталоге и выводит их список. Студент выбирает 4 основных и 2 альтернативных курса. Когда выбор осуществлен, система создает график студента, и выполняется подчиненный поток «Принять график».

Обновить график. Система выводит текущий график студента, выполняет поиск доступных конкретных курсов в каталоге и выводит их список. Студент, в свою очередь, может обновить свой выбор, удаляя или добавляя конкретные курсы, затем происходит обновление графика и выполняется подчиненный поток «Принять график».

Удалить график.

Система выводит текущий график студента и запрашивает у него подтверждения удаления графика. Только после этого график может быть удален, и студент не будет зарегистрирован на выбранных ранее курсах.

Принять график.

Для каждого выбранного, но еще не «зафиксированного» конкретного курса в графике система проверяет выполнение студентом предварительных требований (прохождение определенных курсов), факт открытия конкретного курса и отсутствие конфликтов графика. Затем система добавляет студента в список выбранного конкретного курса. Курс фиксируется в графике и график сохраняется в системе.

Альтернативные потоки

1. Сохранить график.

В любой момент студент может вместо принятия графика сохранить его. В этом случае шаг «Принять график» заменяется на следующий:

«Незафиксированные» конкретные курсы помечаются в графике как «выбранные».

2. График сохраняется в системе.

Не выполнены предварительные требования, курс заполнен или имеют место противоречия в графиках. Если во время выполнения подчиненного потока «Принять график» система обнаружит присутствие указанных условий, будет выдано сообщение об ошибке. Студент может либо выбрать другой конкретный курс и продолжить выполнение варианта использования, либо сохранить график, либо отменить операцию, после чего основной поток начнется заново.

3. График не найден.

Если во время выполнения подчиненных потоков «Обновить график» или «Удалить график» система не может найти график студента, то выдается сообщение об ошибке. Когда студент подтвердит это сообщение, основной поток начнется сначала.

4. Система каталога курсов недоступна.

Если окажется, что невозможно установить связь с системой каталога курсов, то будет выдано сообщение об ошибке. После того как студент подтвердит это сообщение, вариант использования завершится.

5. Регистрация на курсы закончена.

Если в самом начале выполнения варианта использования окажется, что регистрация на текущий семестр закончена, будет выдано сообщение об ошибке и вариант использования завершится.

6. Удаление отменено.

Если во время выполнения подчиненного потока «Удалить график» студент решит не удалять его, удаление отменяется, и основной поток начнется с начала.

Предусловия

Перед началом выполнения данного варианта использования студент должен войти в систему.

Постусловия

Если вариант использования завершится успешно, график студента будет создан, обновлен или удален. В противном случае состояние системы не изменится.

Вариант использования Close Registration.

Краткое описание

Данный вариант использования позволяет регистратору закрывать процесс регистрации. Конкретные курсы, на которые не записалось достаточного количества студентов (менее трех), отменяются. В расчетную систему передается информация о каждом студенте по каждому конкретному курсу, чтобы студенты могли внести оплату за курсы.

Основной поток событий

Данный вариант использования начинает выполняться, когда регистратор запрашивает прекращение регистрации.

Система проверяет состояние процесса регистрации. Если регистрация еще выполняется, выдается сообщение и вариант использования завершается.

Для каждого конкретного курса система проверяет, ведет ли его какой-либо профессор и записалось ли на него не менее трех студентов. Если эти условия выполняются, система фиксирует конкретный курс в каждом включающем его.

Для каждого студенческого графика проверяется наличие максимального количества основных курсов; если их недостаточно, система пытается дополнить альтернативными курсами из списка текущего графика. Выбирается первый доступный альтернативный курс, если его нет, дополнение не происходит.

Система закрывает все конкретные курсы. Если в каком-либо курсе оказывается менее трех студентов (с учетом добавлений, сделанных в пункте 3), система отменяет его и исключает из каждого содержащего его графика.

Система рассчитывает плату за обучение для каждого студента в текущем семестре и направляет информацию в расчетную систему, которая, в свою очередь, посылает студентам счет для оплаты с копией их окончательных графиков.

Альтернативные потоки

Конкретный курс никто не ведет. Если во время выполнения основного потока обнаруживается, что некоторый конкретный не ведется никаким профессором, то этот курс отменяется, и система исключает его из каждого содержащего его графика.

Расчетная система недоступна. Если невозможно установить связь с расчетной системой, через некоторое определенное время система вновь попытается связаться с ней. Попытки будут повторяться до тех пор, пока связь не установится.

Предусловия

Перед началом выполнения данного варианта использования регистратор должен войти в систему.

Постусловия

Если вариант использования завершится успешно, регистрация закрывается. В противном случае состояние системы не изменится.

Прикрепление файла к варианту использования

Щелкнуть правой кнопкой мыши на варианте использования, в открывшемся меню выбрать пункт Properties. Далее перейти на вкладку Files и указать путь к ранее созданному файлу (File Path), нажать на кнопку Save, чтобы прикрепить файл к варианту использования. Для открытия прикрепленного файла нажать кнопку Launch.

Удаление вариантов использования и действующих лиц. Существует два способа удаления элемент модели – из одной диаграммы или из всей модели. Чтобы удалить элемент модели из диаграммы, выделить элемент на диаграмме, нажать на клавишу Delete или сочетание клавиш CTRL + D. Обратить внимание, что, хотя элемент и удален с диаграммы, он остался в браузере и на других диаграммах системы. Чтобы удалить элемент из модели, нужно выделить элемент на диаграмме и выбрать пункт меню Delete.

3.4 Практическая работа № 4. Разработка диаграммы классов

Принятие соглашений по моделированию включает используемые диаграммы и элементы модели, правила их применения, соглашения по именованию элементов и организацию модели (пакеты).

Пример соглашений моделирования:

1. Имена вариантов использования задают короткими глагольными фразами. Для каждого варианта использования необходимо создать пакет Use-Case Realization, включающий, во-первых, по крайней мере одну реализацию варианта использования, во-вторых, диаграмму «View Of Participating Classes» (VOPC).
2. Имена классов должны быть существительными, соответствующими по возможности понятиям предметной области (глоссарию проекта).
3. Имена классов начинают с заглавной буквы.
4. Имена атрибутов и операций начинают со строчной буквы.
5. Составные имена должны быть сплошными, без подчеркиваний, каждое отдельное слово следует начинать с заглавной буквы.

Реализация варианта использования (Use-Case Realization) описывается терминами взаимодействующих объектов и представляется с помощью набора диаграмм (диаграмм классов, реализующих вариант использования, и диаграмм взаимодействия (диаграмм последовательности), отражающих взаимодействие объектов в процессе реализации варианта использования) (рис. П.1.9).

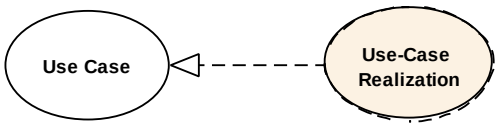


Рис. П.9. Реализация варианта использования

Идентификация ключевых абстракций заключается в предварительном определении классов системы (классов анализа). Источники – знание предметной области, требования к системе, глоссарий. Классы анализа для системы регистрации показаны на рис. П.10:

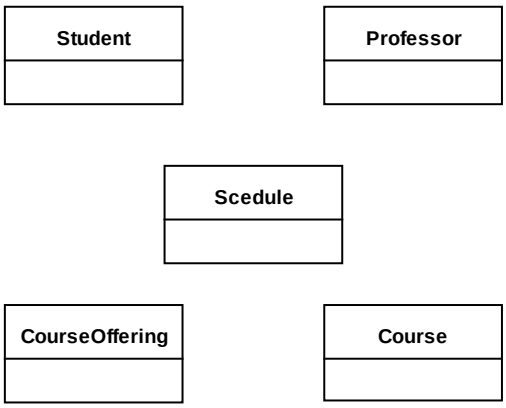


Рис. П.10. Классы анализа системы регистрации

Создание структуры модели и классов анализа в соответствии с требованиями архитектурного анализа

Структура логического представления браузера должна выглядеть так, как показано на рис. П.11.

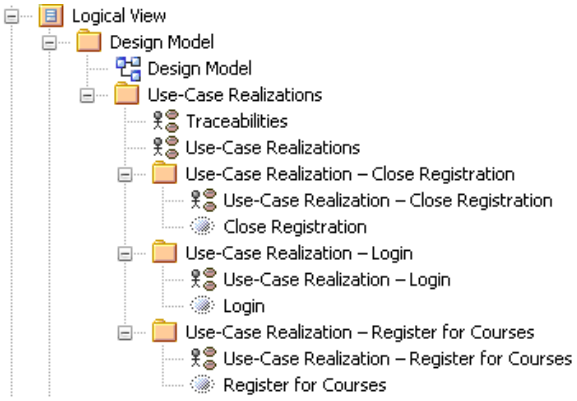


Рис. П.11. Структура логического представления

Для создания пакетов и диаграммы Traceabilities щелкнуть правой кнопкой мыши на логическом представлении браузера. В открывшемся меню выбрать пункт Add > Add Package. Новому пакету присвоить имя Design Model.

Аналогичным образом создать пакеты Use-Case Realizations, Use-Case Realization – Close Registration, Use-Case Realization – Login и Use-Case Realization – Register for Courses.

В каждом из пакетов типа Use-Case Realization сформировать соответствующие кооперации Close Registration, Login и Register for Courses (каждая кооперация представляет собой вариант использования со стереотипом «use-case realization», который задается в спецификации варианта использования).

В пакете Use-Case Realizations создать новую диаграмму вариантов использования с названием Traceabilities и постpить ее в соответствии с рис. П.12.

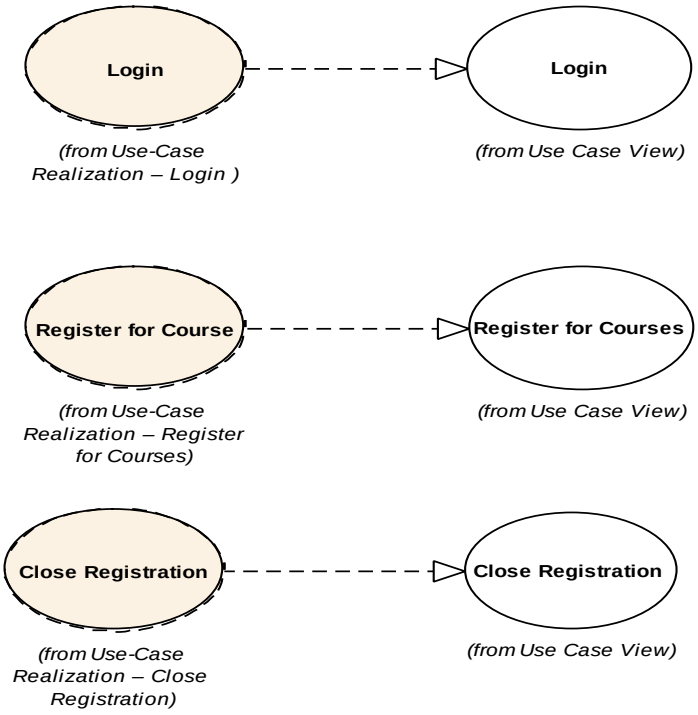


Рис. П.12. Диаграмма Traceabilities

Анализ вариантов использования

Классы, участвующие в реализации потоков событий варианта использования:

граничные (Boundary) – служат посредниками при взаимодействии внешних объектов с системой. Как правило, для каждой пары «действующее лицо – вариант использования» определяется один граничный класс. В свою очередь подразделаются на типы: пользовательский интерфейс (обмен информацией с пользователем, без деталей интерфейса – кнопок, списков, окон), системный интерфейс и аппаратный интерфейс (используемые протоколы, без деталей их реализации);

классы-сущности (Entity) – представляют собой ключевые абстракции (понятия) разрабатываемой системы. Источники выявления: ключевые абстракции, созданные в процессе архитектурного анализа, глоссарий, описание потоков событий вариантов использования;

управляющие (Control) – обеспечивают координацию поведения объектов в системе. Могут отсутствовать в некоторых вариантах использования, ограничивающихся простыми манипуляциями с хранимыми данными. Как правило, для каждого варианта использования определяется один управляющий класс. К этому типу можно отнести менеджер транзакций и обработчик ошибок.

Пример набора классов, участвующих в реализации варианта использования Register for Courses, приведен на рис. П.13.

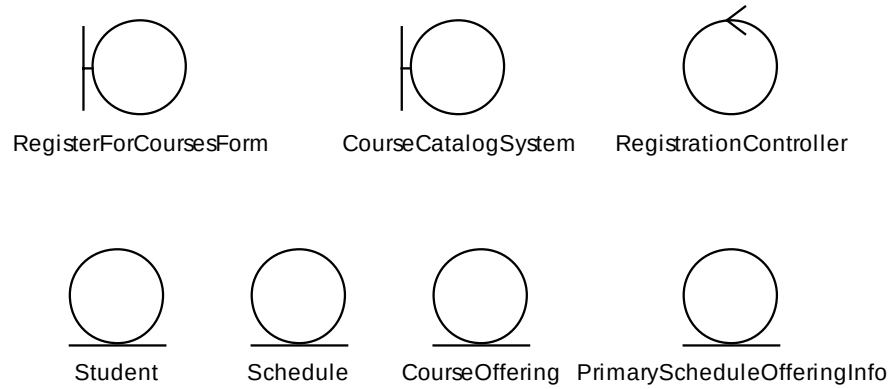


Рис. П.13. Классы, участвующие в реализации варианта использования Register for Courses

Создание классов, участвующих в реализации варианта использования Register for Courses, и диаграммы классов «View Of Participating Classes» (VOPC)

Щелкнуть правой кнопкой мыши на пакете Design Model, выбрать в открывшемся меню пункт Add > Add Element, выберите тип Class. Ввести имя RegisterForCoursesForm.

В открывшемся окне в поле стереотипа выбрать Boundary и нажать на кнопку ОК. Создать аналогичным образом классы CourseCatalogSystem со стереотипом Boundary и RegistrationController со стереотипом Control. Назначить классам Schedule, CourseOffering и Student стереотип Entity.

Щелкнуть правой кнопкой мыши на кооперации Register for Courses в пакете Use-Case Realization – Register for Courses, в открывшемся меню выбрать пункт Add > Add Diagram > Class. Новую диаграмму классов назвать VOPC (classes only), открыть ее и перетащить на нее классы в соответствии с рис. П.13.

Распределение поведения, реализуемого вариантом использования, между классами описывается с помощью диаграмм взаимодействия (диаграмм последовательности и кооперативных диаграмм). В первую очередь строится диаграмма (одна или более), описывающая основной поток событий и его подчиненные потоки. Для каждого альтернативного потока событий строится отдельная диаграмма.

В качестве примеров можно привести обработку ошибок, контроль времени выполнения и обработку неправильно вводимых данных. Описывать тривиальные потоки событий (например, в потоке участвует только один объект) нецелесообразно.

3.5 Практическая работа № 5. Разработка диаграммы последовательности и редактирование свойств ее элементов

Создание диаграмм взаимодействия

Создать диаграммы последовательности и кооперативные диаграммы для основного потока событий варианта использования Register for Courses. Они должны иметь вид, как на рис. П.14 – П.18.

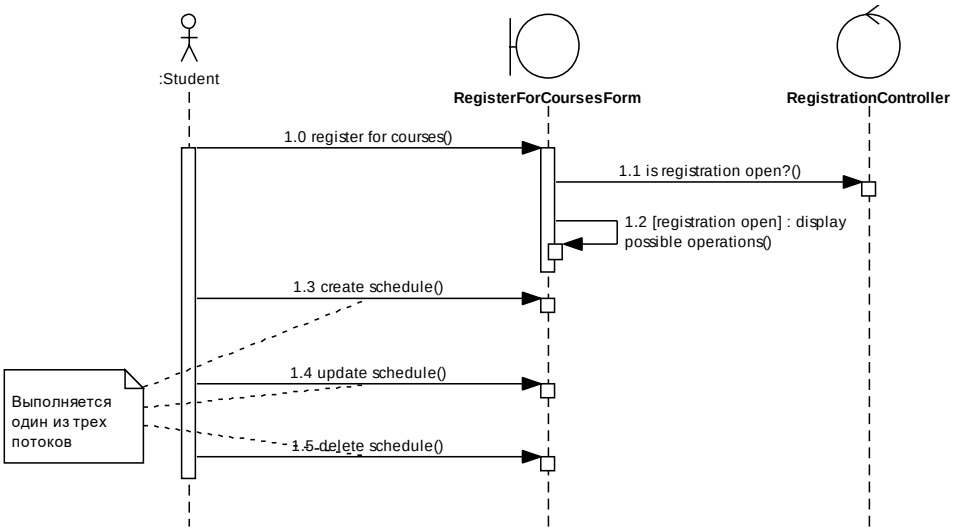


Рис. П.14. Диаграмма последовательности Register for Courses – Basic Flow

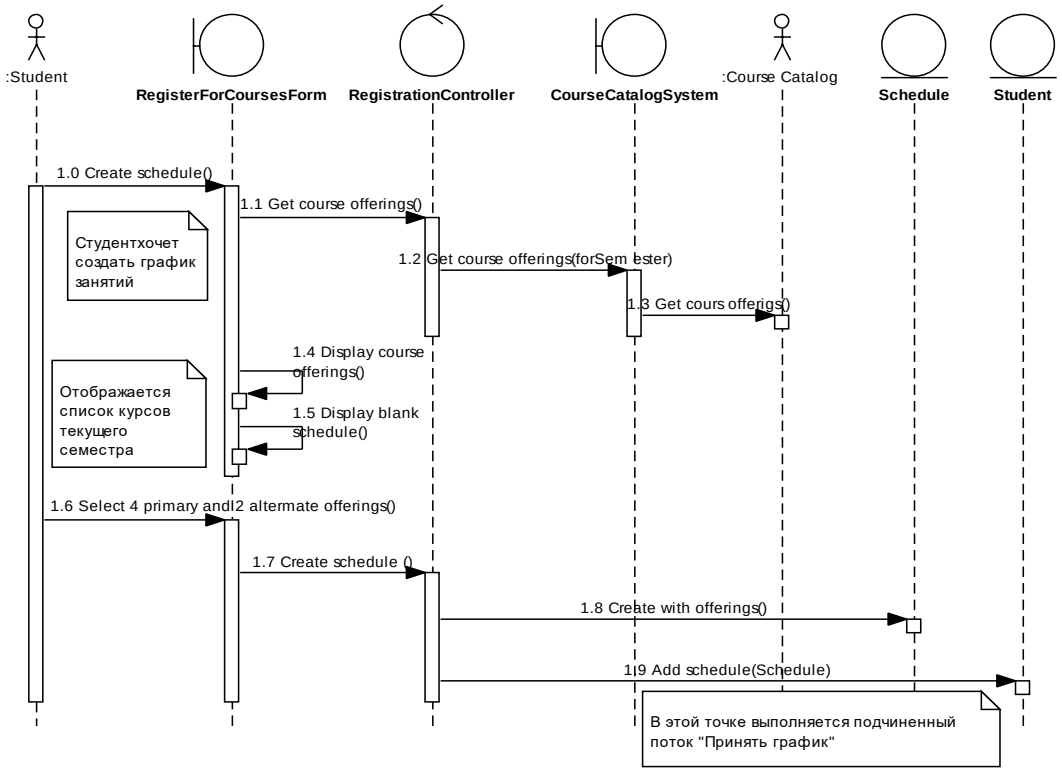


Рис. П.15. Диаграмма последовательности Registerfor Courses – Basic Flow (Create Schedule)

Настройка

В меню модели выбрать пункт Tools > Options или нажать сочетание клавиш CTRL + F9. Перейти на вкладку Diagram > Sequence. Контрольный переключатель Show Sequence Numbering должен быть помечен. Чтобы выйти из окна параметров, нажать Close.

Создание диаграммы последовательности

Щелкнуть правой кнопкой мыши на кооперации Register for Courses в пакете Use-Case Realization – Register for Courses, в открывшемся меню выбрать пункт Add > Add Diagram > Sequence и назвать новую диаграмму Register for Courses – Basic Flow. Открыть ее, дважды щелкнув на ней.

Затем добавить на диаграмму действующее лицо, объекты и сообщения. Для этого перетащить действующее лицо Student из браузера на диаграмму, то же самое проделать с классами RegisterForCoursesForm и RegistrationController. На панели инструментов нажать кнопку Message (Сообщение). Провести мышью от линии жизни действующего лица Student к линии жизни объекта RegisterForCoursesForm. Выделив сообщение, ввести его имя: register for courses.

Повторить описанные выше действия, чтобы поместить на диаграмму остальные сообщения, как показано на рис. П.14 (для рефлексивного сообщения 1.2 используется кнопка Self -Message).

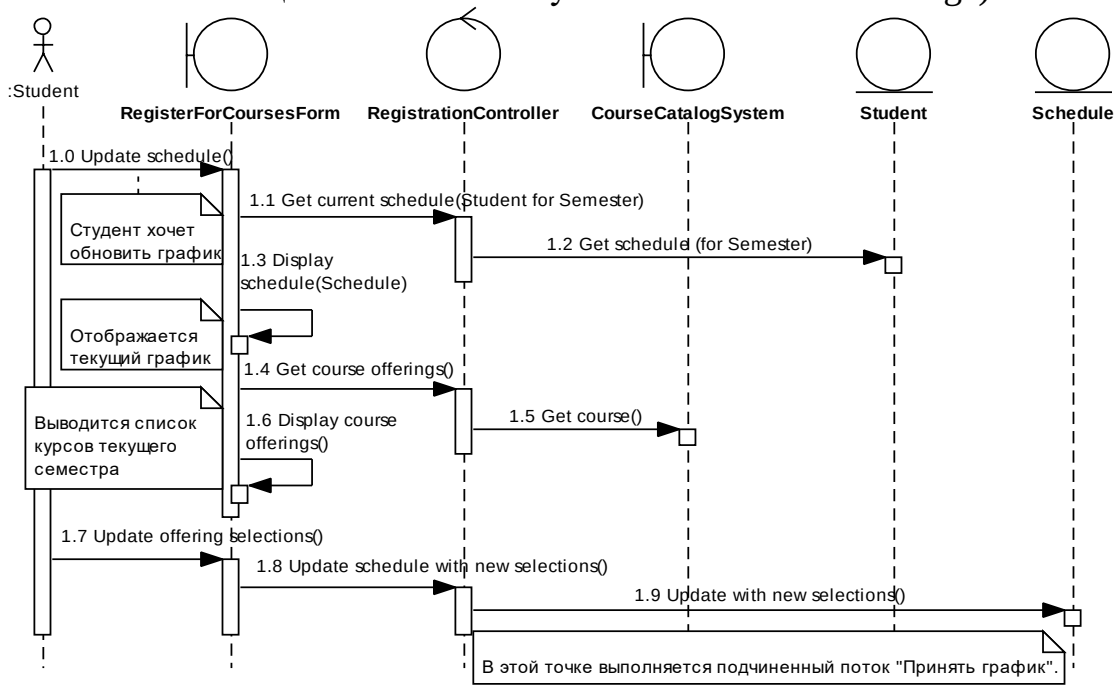


Рис. П.16. Диаграмма последовательности Register for Courses – Basic Flow (Update Schedule)

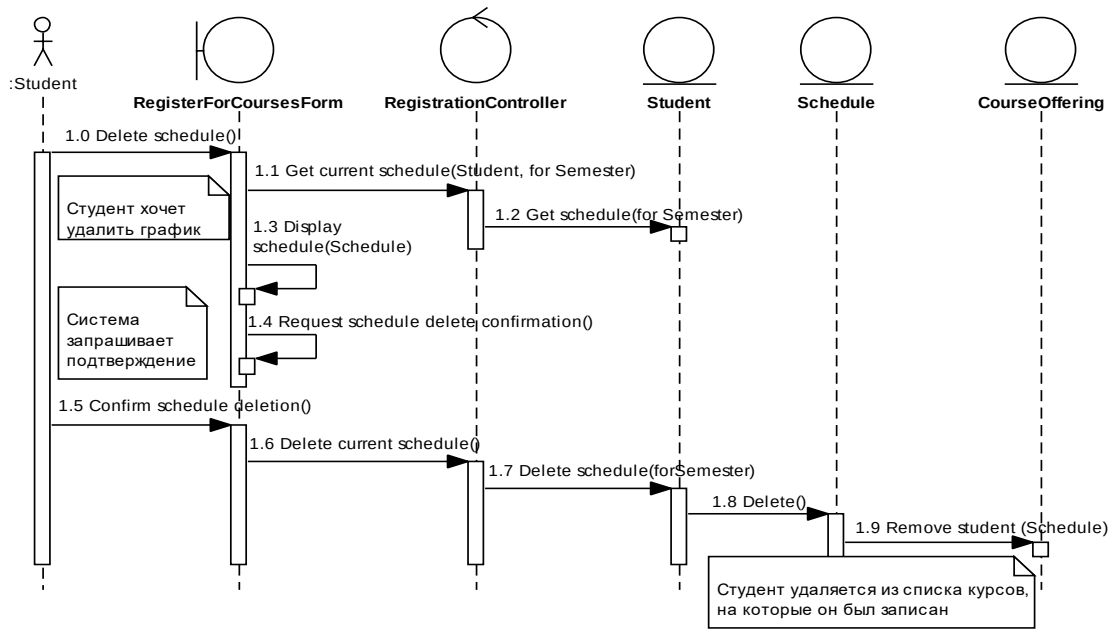


Рис. П.17. Диаграмма последовательности Register for Courses – Basic Flow (Delete Schedule)

Выполнить аналогичные действия для создания диаграмм последовательности, показанных на рис. П.11–П.18. При этом обратить внимание, что на диаграмме (рис. П.18) появился объект нового класса PrimarySheduleOffering-Info (класса ассоциаций, описывающего связь между классами Shedule и OfferingInfo), который нужно предварительно создать.

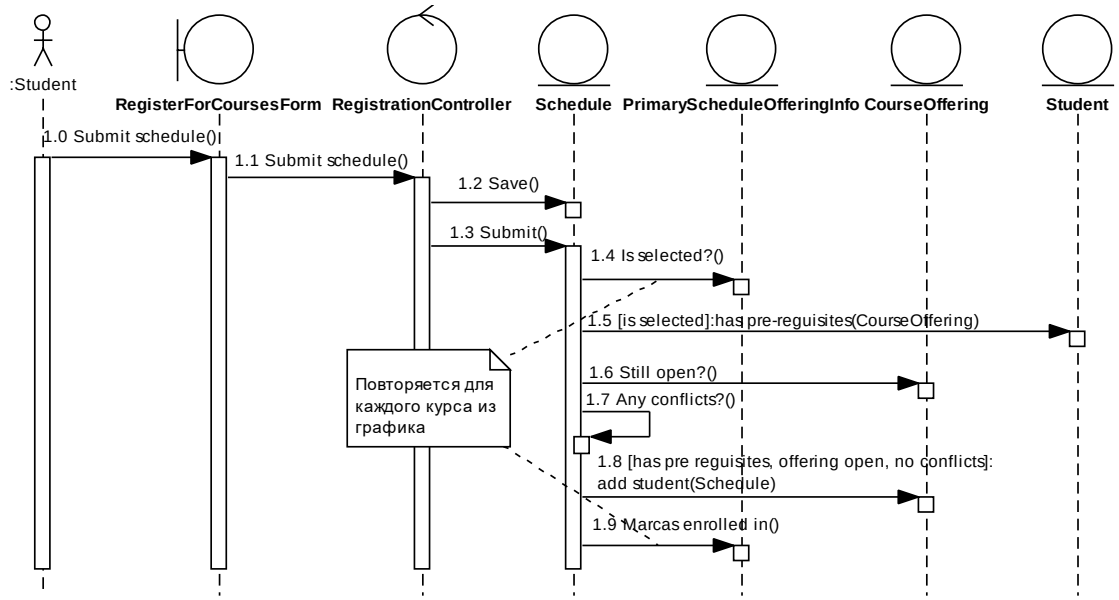


Рис. П.18. Диаграмма последовательности Register for Courses – Basic Flow (Submit Schedule)

Создание примечаний:

Нажать на панели инструментов кнопку Note и щелкнуть мышью в том месте диаграммы, где предполагается разместить примечание. Выделть новое примечание и ввести в него текст.

Чтобы прикрепить примечание к элементу диаграммы, щелкнуть по примечанию правой кнопкой мыши и в появившемся окне выбрать Advanced > Set Attached Links. Установить флажок напротив того (тех) элемента (элементов) диаграммы, с которым будет связано примечание. Между примечанием и элементом возникнет штриховая линия.

Поместить на диаграмму текстовую область. Для этого на панели управления нажать кнопку New Text Element, щелкнуть мышью внутри диаграммы, чтобы поместить туда текстовую область, и ввести в нее текст.

3.6 Практическая работа № 6. Разработка диаграммы классов на уровне сущностей

Диаграммы последовательности позволяют сформировать все операции (методы) классов. Вид классов после добавления операций представлен на рис. П.19.

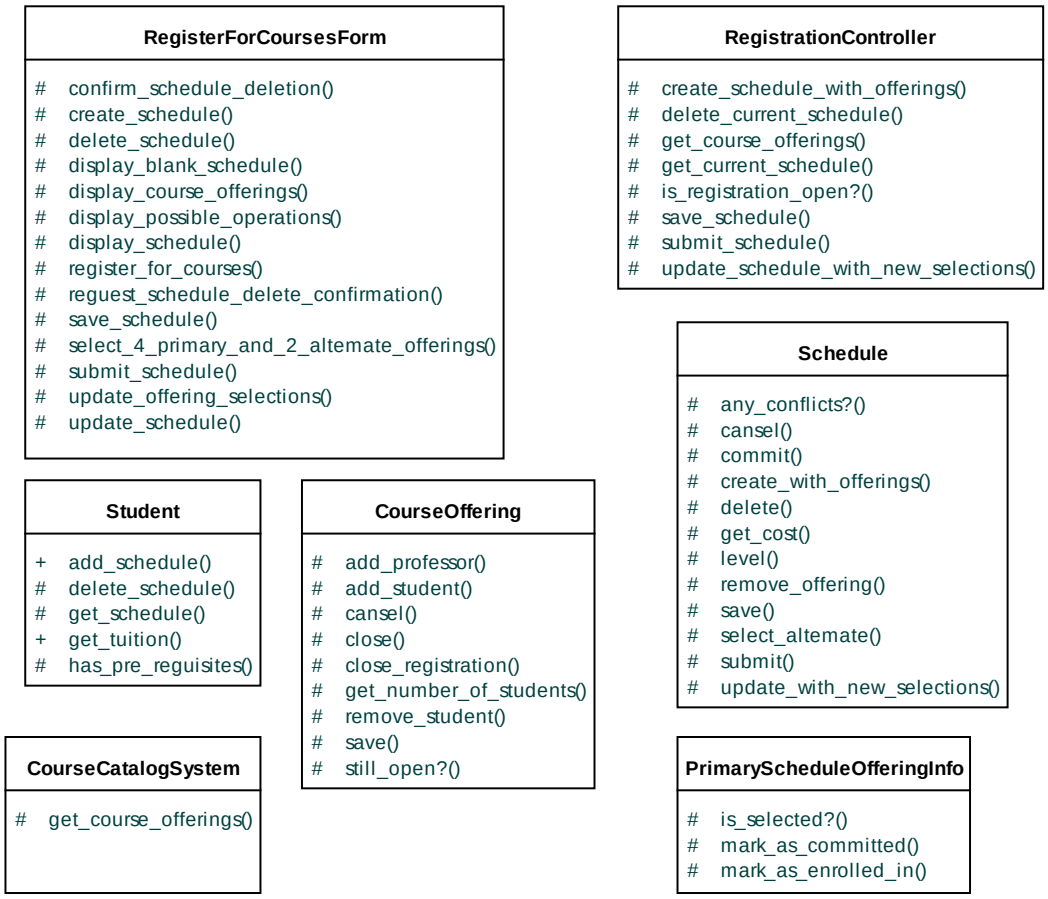


Рис. П.19. Диаграмма классов с операциями

Проектирование баз данных

Разработка таблиц базы данных

Создание новой таблицы:

нажать на панели инструментов кнопку Table;

щелкнуть мышью в том месте диаграммы, куда нужно поместить таблицу;

в открывшемся окне ввести имя таблицы;

в окне Database выбрать тип базы данных.

Для добавления колонок таблицы и редактирования их свойств:

щелкнуть правой кнопкой по таблице;

в открывшемся меню выбрать пункт Attributes;

в появившемся окне ввести название колонки;

в окне Data Type выберите тип данных, в окне Initial при необходимости ввести начальное значение, которое может быть использовано как значение по умолчанию для колонки.

Если колонка представляет первичный ключ для таблицы, установитть флажок Primary Key. Готовая модель БД изображена на рис. П.20.

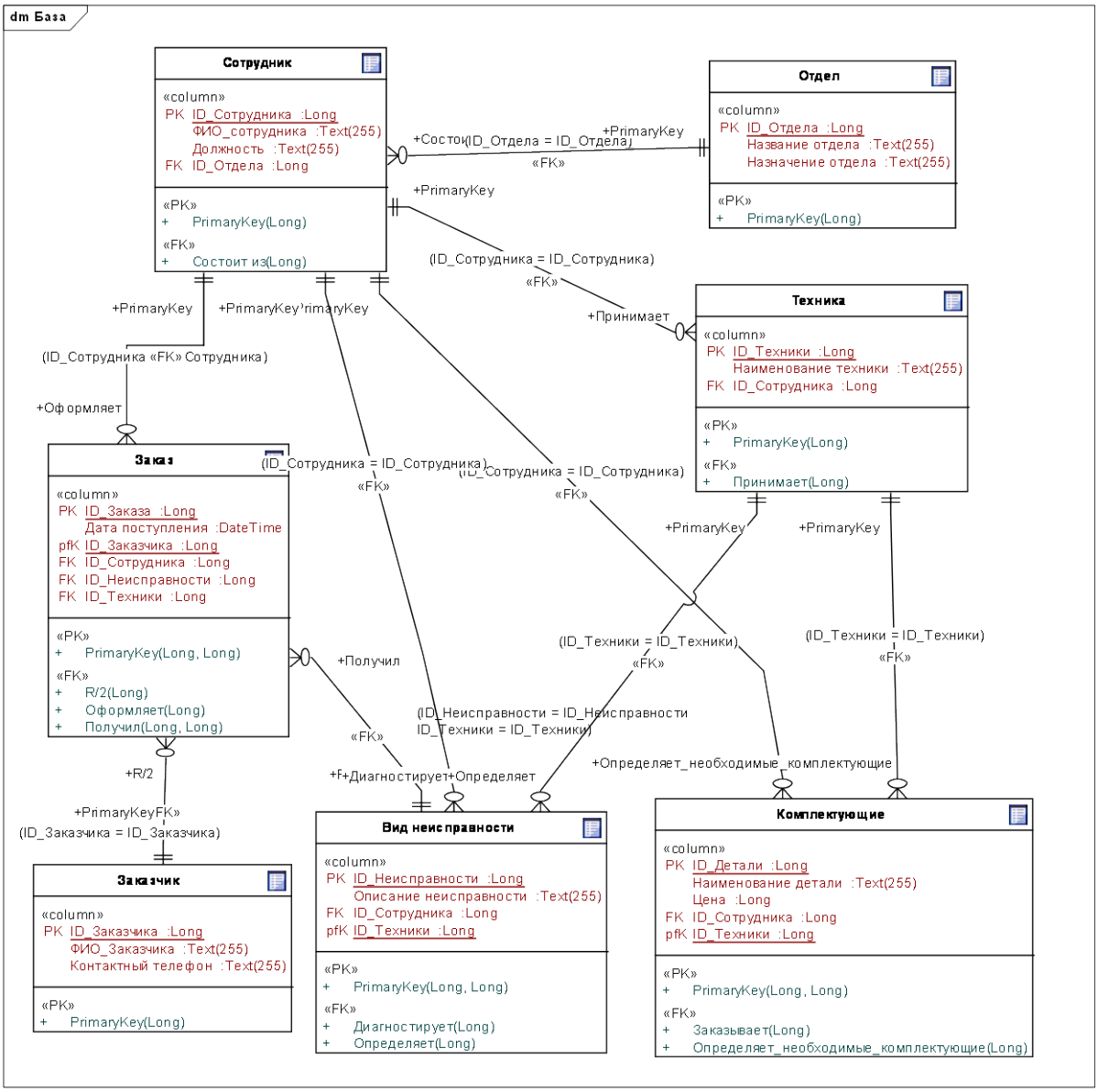


Рис. П.20. Вид модели данных в Enterprise Architect

Enterprise Architect может генерировать простые DDL-скрипты для создания таблиц в разрабатываемой модели.

Генерация DDL для таблицы:

щелкнуть правой кнопкой мыши по таблице, для которой генерируется DDL-скрипт;

в появившемся меню выбрать пункт Generate DDL;

в открывшемся окне выбрать путь, куда будет сгенерирован скрипт;

при необходимости установить флажки;

для создания DDL-скрипта нажать кнопку Generate, когда генерация завершится, открыть созданный текстовый файл и просмотреть результаты.

Добавление атрибутов

щелкнуть правой кнопкой мыши на классе Student;

в открывшемся меню выбрать пункт Attributes;

ввести новый атрибут address;

нажать кнопку Save, а для добавления нового атрибута – кнопку New;

добавить атрибуты к классам CourseOffering, Shedule и PrimarySchedule-OfferingInfo, как показано на рис. П.21.

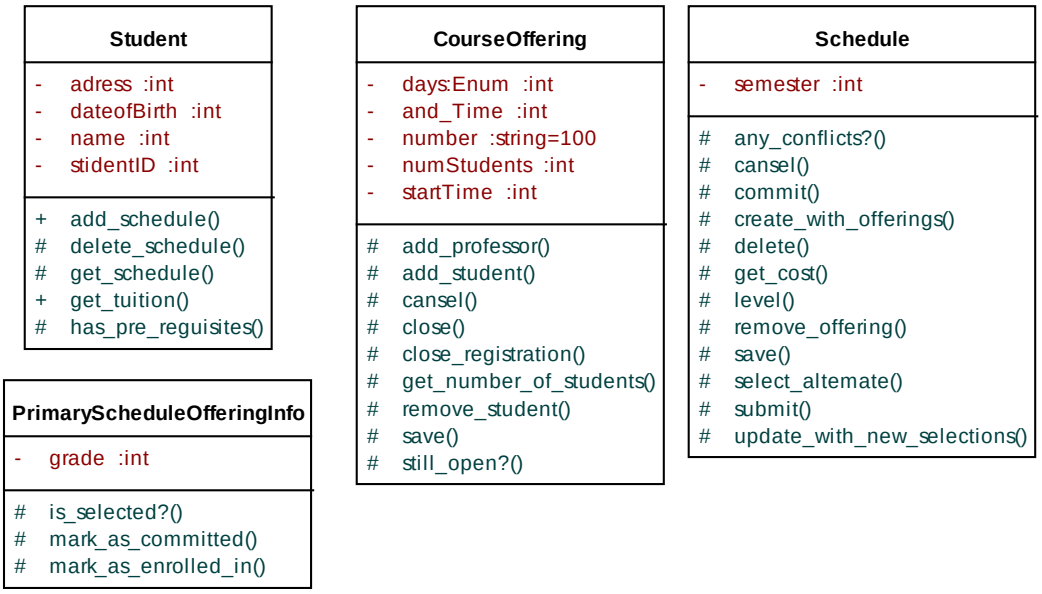


Рис. П.21. Классы с операциями и атрибутами

Связи между классами (ассоциации) определяются на основе диаграмм взаимодействия. Если два объекта взаимодействуют (обмениваются сообщениями), между ними должна существовать связь (путь взаимодействия). Для ассоциаций задаются множественность и, возможно, направление навигации. Могут использоваться множественные ассоциации, агрегации и классы ассоциаций.

Добавление связей

Добавим связи к классам, принимающим участие в варианте использования Register for Courses. Для отображения связей между классами построим три новых диаграмм классов в кооперации Register for Courses пакета Use-Case Realization – Register for Courses (рис. П.22–П.24).

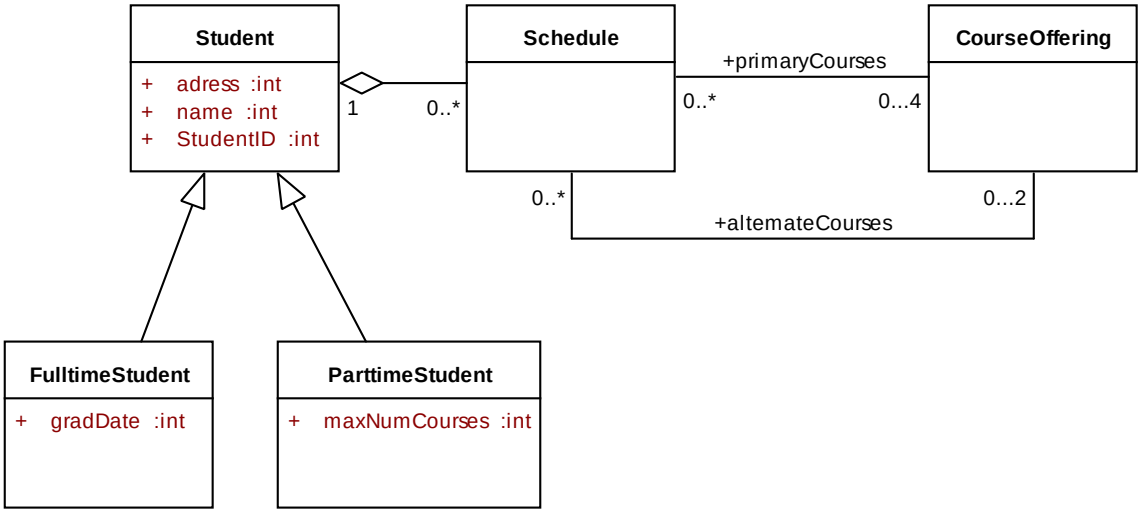


Рис. П.22. Диаграмма Entity Classes (классы-сущности)

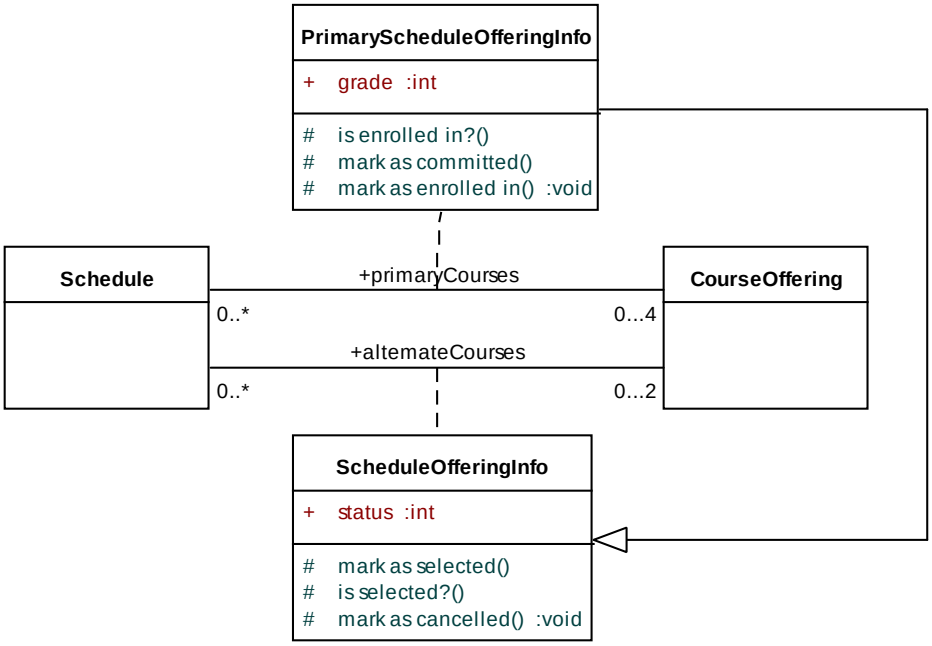


Рис. П.23. Диаграмма CourseOfferingInfo

Добавлены два новых класса – подклассы FulltimeStudent (студент очного отделения) и ParttimeStudent (студент вечернего отделения).

На диаграмме показаны классы ассоциаций, описывающие связи между классами Schedule и CourseOffering и добавлен суперкласс ScheduleOfferingInfo. Данные и операции, содержащиеся в этом классе (status – курс включен в график или отменен), относятся как к основным, так и к альтернативным курсам, в то время как оценка (grade) и окончательное включение курса в график могут иметь место только для основных курсов.

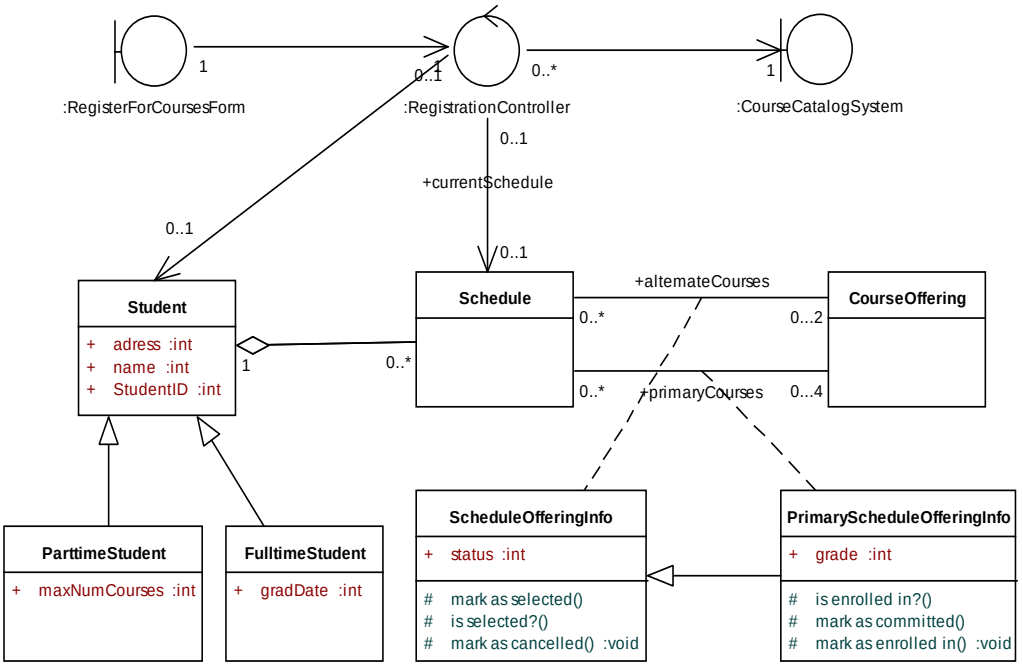


Рис. П.24. Полная диаграмма классов VOPC

Создание ассоциаций

Ассоциации создают непосредственно на диаграмме классов. Панель инструментов диаграммы классов содержит кнопки для создания как одно-, так и двунаправленных ассоциаций. Чтобы на диаграмме классов создать ассоциацию, нужно нажать на панели инструментов кнопку Association Class и провести мышью линию ассоциации от одного класса к другому.

Создание агрегаций

Нажать кнопку Aggregate панели инструментов и провести линию агрегации от класса-части к целому.

Создание обобщений

При создании обобщения может потребоваться перенести некоторые атрибуты или операции из одного класса в другой. Чтобы поместить обобщение на диаграмму классов, нужно нажать кнопку Generalize панели инструментов и провести линию обобщения от подкласса к суперклассу.

Для задания множественности связи щелкнуть правой кнопкой мыши на одном конце связи, в открывшемся меню выбрать пункт Multiplicity, затем указать нужную множественность. То же самое повтроить для другого конца связи. Далее выделить нужную связь и ввести ее имя.

Чтобы задать связи ролевое имя, щелкнуть правой кнопкой мыши на ассоциации с нужного конца, в открывшемся меню выбрать пункт Target Role и ввести ролевое имя.

Классы анализа преобразуются в *проектные классы:*

проектирование граничных классов зависит от возможностей среды разработки пользовательского интерфейса;

проектирование классов-сущностей осуществляется с учетом соображений производительности (выделение в отдельные классы атрибутов с различной частотой использования);

проектирование управляющих классов – удаление классов, реализующих простую передачу информации от граничных классов к сущностям;

идентификация устойчивых (persistent) классов, содержащих хранимую информацию.

Обязанности классов, определенные в процессе анализа, преобразуются в операции. Каждой операции присваивается имя, характеризующее ее результат. Создается краткое описание операции, включая смысл всех ее параметров. Определяется видимость операции: public, private, protected.

Определяются (уточняются) атрибуты классов:

кроме имени, задается тип и значение по умолчанию (необязательное);

учитываются соглашения по именованию атрибутов, принятые в проекте и языке реализации;

задается видимость атрибутов: public, private, protected;

при необходимости определяются производные (вычисляемые) атрибуты.

Определение атрибутов и операций для класса Student

Чтобы задать тип данных, значение по умолчанию и видимость атрибута, щелкнуть правой кнопкой мыши на классе в браузере, в открывшемся меню выбрать пункт Attributes. В раскрывающемся списке типов указать тип данных или задать его самостоятельно. В поле Initial (Первоначальное значение) ввести значение атрибута по умолчанию. В поле Scope выбрать видимость атрибута: Public, Protected или Private (по умолчанию видимость всех атрибутов соответствует Private). Получившийся класс представлен на рис. П.25.

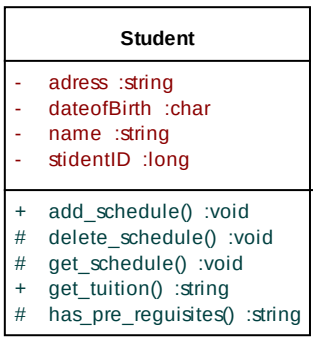


Рис. П.25. Класс Student с полностью определенными операциями и атрибутами

Чтобы задать тип возвращаемого значения, стереотип и видимость операции, щелкнуть правой кнопкой мыши на операции в браузере и в появившемся окне выбрать Operations. Указать тип возвращаемого значения (Return Type) в раскрывающемся списке или ввести свой тип.

В соответствующем раскрывающемся списке указать стереотип или ввести новый. В поле Scope отметить значение видимости операции: Public, Protected или Private (по умолчанию видимость всех операций установлена в Public). Затем добавить к операции параметр. Для этого открыть окно спецификации операции, нажать кнопку «Edit Parameters», ввести имя параметра и выбрать тип данных аргумента. Если требуется, ввести значение аргумента по умолчанию (Default). Нажав кнопку «Save», сохранить параметр.

3.7 Практическая работа № 7. Разработка диаграммы состояний и редактирование свойств ее элементов

Цель работы: изучить технологии создания диаграммы состояний.

Определение состояний для классов моделируется с помощью диаграмм состояний, которые создаются для описания объектов с высоким уровнем динамического поведения (рис. П.26).

В качестве примера рассмотрим поведение объекта класса CourseOffering. Он может находиться в открытом (возможно добавление нового студента) или закрытом состоянии (максимальное количество студентов уже записалось на курс). Таким образом, конкретное состояние зависит от количества студентов, связанных с объектом CourseOffering. Рассматривая каждый вариант использования, можно выделить еще два состояния: инициализация (до начала регистрации студентов на курс) и отмена (курс исключается из расписания).

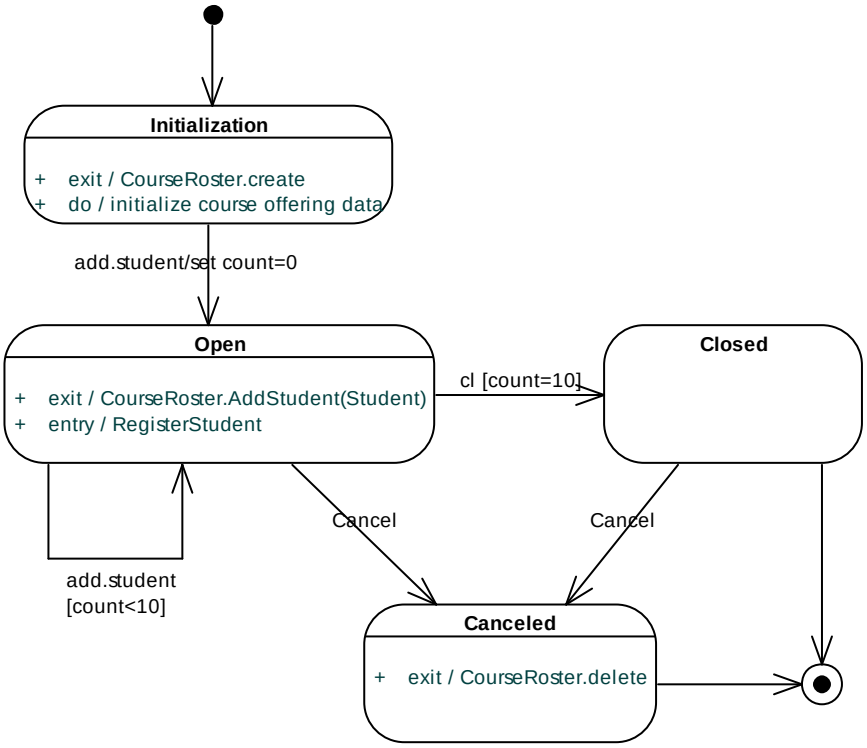


Рис. П.26. Диаграмма состояний для класса CourseOffering

Создание диаграммы состояний для класса Course-Offering

Для создания диаграммы состояний щелкнуть правой кнопкой мыши в браузере на нужном классе и в открывшемся меню выбрать пункт Add > Add Diagram > State Machine.

Чтобы добавить состояние, на панели инструментов нажать кнопку State, затем щелкнуть мышью на диаграмме состояний в том месте, куда нужно его поместить.

Для добаавления деятельности щелкнуть правой кнопкой мыши по требуемому состоянию, в открывшемся меню выбрать пункт Operations., затем в поле Name ввести название деятельности. Чтобы новое действие стало деятельностью, в окне Action указать do. Если нужно добавить вхожное действие, в окне Action указать entry, если выходное – exit.

Чтобы создать переход, нажать кнопку Transition панели инструментов, затем нажать левую клавишу мыши на состоянии, откуда осуществляется переход. Не отпуская клавишу мыши, провести линию перехода до того состояния, где он завершается.

Для добавления рефлексивного перехода нажать кнопку Transition на панели инструментов и щелкнуть на том состоянии, где осуществляется рефлексивный переход.

Чтобы добавить событие, ограждающее условие и действие, дважды щелкнуть на переходе. Откроется окно его спецификации. Далее на вкладке General ввести событие в поле Name, перейти на вкладку

Constraints и ввести ограждающее условие в поле Guard. Затем в поле Effect ввести действие.

Для указания начального или конечного состояния на панели инструментов нажать кнопку Initial или Final. Щелкнуть мышью на диаграмме состояний в том месте, куда нужное поместить состояние.

3.8 Практическая работа № 8. Разработка диаграммы компонентов и редактирование свойств ее элементов

В Enterprise Architect диаграммы компонентов создаются в представлении компонентов системы. Отдельные компоненты можно создавать непосредственно на диаграмме или перетаскивать их туда из браузера.

Создание диаграммы компонентов

Диаграмма компонентов показывает части программного обеспечения, встроенных контроллеров, их организацию и зависимость. Она имеет более высокий уровень абстракции, чем диаграмма класса, обычно компонент осуществляется одним или более классами (или объектами) во время выполнения. Они являются структурными элементами, построенными так, что, в конечном счете, один из компонентов может охватить значительную часть системы.

Компонент является модульной частью системы, его поведение определяется provided и required интерфейсами; внутреннее функционирование компонента должно быть невидимым, а его использование независимым от окружающей среды.

Создание компонента:

дважды щелкнуть мышью на главной диаграмме компонентов в представлении компонентов;

нажать на панели инструментов кнопку Component;

щелкнуть мышью в том месте диаграммы, куда будет помещен компонент.

в открывшемся окне ввести название компонента, выбрать его стереотип, язык программирования.

3.9 Практическая работа № 9. Разработка диаграммы размещения и редактирование свойств ее элементов

Распределенная конфигурация системы моделируется с помощью диаграммы размещения. Ее основные элементы:

узел (node) – вычислительный ресурс (процессор или другое устройство (дисковая память, контроллеры различных устройств и т. д.). Для узла можно задать выполняющиеся на нем процессы;

соединение (connection) – канал взаимодействия узлов (сеть). Пример: сетевая конфигурация системы регистрации (рис. П.27).

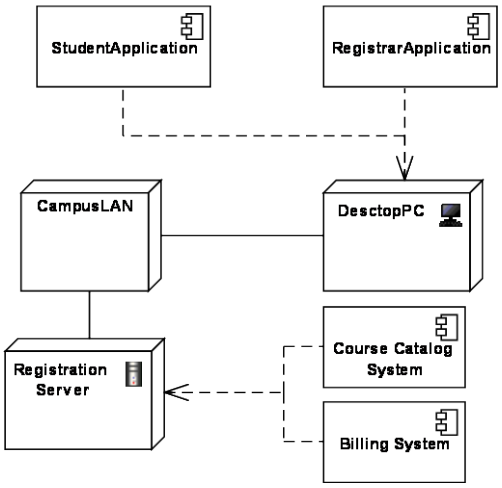


Рис. П.27. Сетевая конфигурация системы регистрации

Распределение процессов по узлам сети производится с учетом следующих факторов:

используемые образцы распределения (трехзвенная клиент-серверная конфигурация, «толстый клиент», «тонкий клиент», равноправные узлы (peer-to-peer) и т. д.);

время отклика;

минимизация сетевого трафика;

мощность узла;

надежность оборудования и коммуникаций.

Создание диаграммы размещения системы регистрации

Для открытия диаграммы размещения дважды щелкнуть мышью на представлении Deployment View (представлении размещения) в браузере.

Чтобы поместить на диаграмму процессор:

на панели инструментов диаграммы нажать кнопку Node;

щелкнуть на диаграмме размещения в том месте, где он будет помещен;

ввести имя процессора.

В спецификациях процессора можно ввести информацию о его стереотипе и характеристиках. Стереотипы применяются для классификации процессоров, характеристики процессора – это его физическое описание. Оно может, в частности, включать скорость процессора и объем памяти.

Чтобы назначить процессору стереотип:

щелкнуть правой кнопкой по изображению процессора, затем в отрывшемся окне выбрать пункт Properties. На вкладке General выбрать стереотип в поле Stereotype.

3.10 Практическая работа № 10. Генерация кода

Цель работы: изучить влияние диаграммы классов на генерирование программного кода.

Механизм генерации исходного кода предполагает формирование эквивалента исходного кода на основе классов или интерфейсов модели для последующей разработки и компиляции.

Перед генерацией кода убедиться, что свойства по умолчанию для нее соответствует заданным требованиям. Эти свойства расположены в меню Tools > Options > Source Code Engineering. Настройки, которые можно задать по умолчанию, включают конструкторы и деструкторы, методы интерфейсов и варианты Unicode для выбранного языка.

Во время генерации кода Enterprise Architect выбирает информацию из логического и компонентного представлений модели и генерирует большой объем «скелетного» (skeletal) кода:

Классы. Генерируются все классы модели.

Атрибуты. Код включает атрибуты каждого класса, в том числе видимость, тип данных и значение по умолчанию.

Сигнатуры операций. Код содержит определения операций со всеми параметрами, типами данных параметров и типом возвращаемого значения операции.

Связи. Некоторые из связей модели вызывают создание атрибутов при генерации кода.

Компоненты. Каждый компонент реализуется в виде соответствующего файла с исходным кодом.

Например, результат генерации кода C++ класса Student:

```
#include «Schedule.cls»
#include «CourseOffering.cls»
class Student
{
public:
     Student();
     virtual ~Student();
     Schedule *m_Schedule;

     add_schedule(Schedule theSchedule);
     delete_schedule(Semester forSemester);
     Schedule get_schedule(Semester forSemester);
     Double get_tuition();
     Boolean has_pre_requisites(CourseOffering
forCourseOffering);
private:
     String address;
     String name;
     Integer studentID;
```

```
        Date dateofBirth;
};
```

Для генерации кода группы классов:

выделить группу классов на диаграмме;

щелкнуть правой кнопкой мыши по элементу группы и в контекстном меню выбрать Generate Code > Generate Selected elements. В окне Batch Generation будет виден процесс выполнения генерации (см. рис. П.28).

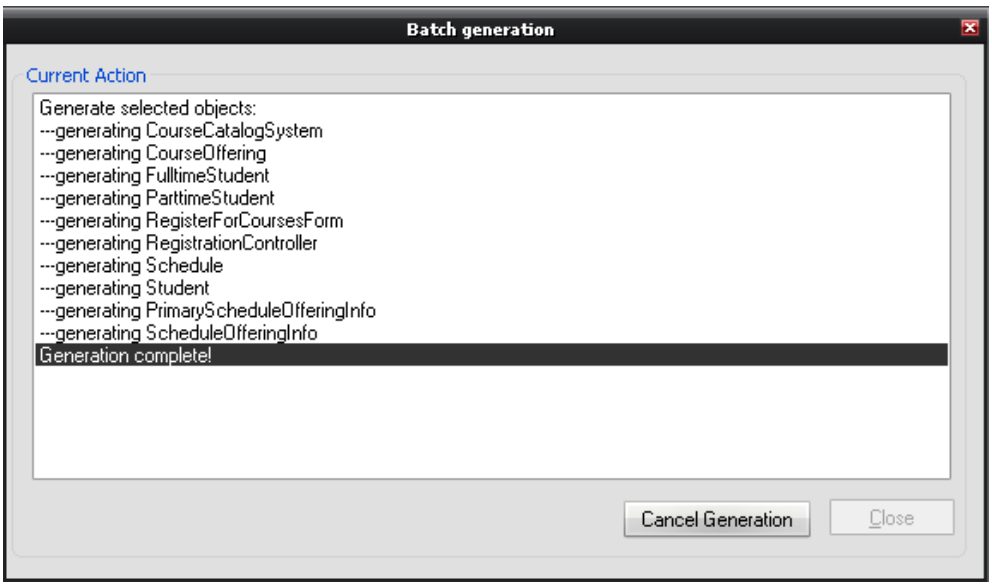


Рис. П.28. Процесс генерации кода группы классов

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Шеер, Август-Вильгельм. ARIS-моделирование бизнес-процессов / Август-Вильгельм Шеер. М.: Вильямс, 2000. 175 с.

2. Биберштейн, Н. Компас в мире сервис-ориентированной архитектуры (SOA): ценность для бизнеса, планирование и план развития предприятия / Н. Биберштейн [и др.]. М.: КУДИЦ-Пресс, 2007. 256 с.

3. Брауде, Э. Технология разработки программного обеспечения / Э. Брауде; пер. с англ. Спб.: Питер. 2004.

4. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++ / Г. Буч. 2-е изд. М.: Бином, СПб.: Невский диалект 1998. 560 с.

5. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Джекобсон. М.: ДМК, 2000. 432 с.

6. Вендров, А.М. Проектирование программного обеспечения экономических информационных систем / А.М. Вендров. М.: Финансы и статистика, 2000. 352 с.

7. Верников, Г. Основы методологии IDEF1, IDEF1X, IDEF3, IDEF5 [Электронный ресурс] / Г. Верников. Режим доступа: http://www.citforum.ru/cfin/vernikov.

8. Верников, Г. Основные методологии обследования организации. Стандарт IDEF0 [Электронный ресурс] / Г. Верников. Режим доступа: http:// www.consulting.ru/main/mgmt/texts/m7/079_idef.shtml.

9. Вигерс, К. Разработка требований к программному обеспечению / К. Вигерс; пер. с англ. М.: Издательско-торговый дом «Русская Редакция», 2004. 576 с.: ил.

10. Гецци, К. Основы инженерии программного обеспечения / К.Гецци, М. Джазайери, Д. Мандриоли; пер. с англ. СПб. БХВ-Петербург. 2005.

11. Дин, Д. Принципы работы с требованиями к программному обеспечению. Унифицированный подход / Д. Дин, У. Дон. М.: Вильямс. 2002.

12. Калянов, Г. CASE: все только начинается... [Электронный ресурс]. Режим доступа: http://www.osp.ru/cio/2001/03/016.htm.

13. Кватрани, Т. Rational Rose 2000 и UML. Визуальное моделирование / Т. Кватрани. М.: ДМК Пресс, 2001. 176 с.

14. Коберн, А. Современные методы описания функциональных требований к системам / А. Коберн. М.: ЛОРИ, 2002.

15. Кознов, Д.В. Визуальное моделирование: теория и практика [Электронный ресурс] / Д.В. Кознов. Режим доступа: http://www.intuit.ru/department/se/vismodtp/3/.

16. Кознов Д.В. Языки визуального моделирования. Проектирование и визуализация программного обеспечения: учеб. пособие / Д.В.Кознов. СПб.: Изд-во С.-Петерб. ун-та, 2004. 171 с.

17. Корпорация: языки управления бизнес-процессами. BPML [Электронный ресурс]. Режим доступа: http://citforum.ru/internet/xml/bpml/.

18. Лаврищева, Е.М. Методы и средства инженерии программного обеспечения / Е.М. Лаврищева, В.А. Петрухин. М., 2006 г.

19. Ларман, Крег. Применение UML и шаблонов проектирования / Крег Ларман. М.: Вильямс, 2001. 496 с.

20. Леоненков, А.В. Объектно-ориентированный анализ и проектирование с использованием UML [Электронный ресурс] / А.В.Леоненков. Режим доступа: www.intuit.ru.

21. Леоненков, А.В. Самоучитель UML 2 / А.В. Леоненков. СПб.: БХВ-Петербург, 2007. 576 с.

22. Маклаков, С.В. Создание информационных систем с AllFusion Modeling Suite / С.В. Маклаков. М.: Диалог-МИФИ, 2007. 400 с.

23. Мацяшек, Л.А. Анализ требований и проектирование систем: разработка информационных систем с использованием UML / Л.А. Мацяшек. М.: Вильямс, 2002. 432 с.

24. Машков, Д.А. Моделирование бизнес-процессов при проектировании КИС [Электронный ресурс] / Д.А. Машков. Режим доступа: http:// www.soft.implozia.ru/news/seminar02_04.html.

25. Маторин, С.И. Моделирование организационных систем в свете нового подхода «Узел-Функция-Объект» / С.И. Маторин, А.С. Попов, В.С.Маторин // Научно-техническая информация. Сер. 2. 2005. № 1.с. 1–8.

26. Маторин, С.И. Теория систем и системный анализ: учебное пособие / С.И. Маторин., О.А. Зимовец. Белгород: Изд-во НИУ «БелГУ», 2012.

27. Михеев, А. Война стандартов в мире workfiow [Электронный ресурс] / А. Михеев, М. Орлов // PC Week/RE. 2004. № 28. Режим доступа: http://wf.runa.ru/rus/images/c/ce/Stat_ya2.pdf.

28. Михеев, А. Перспективы WorkFlow систем. Сравнение WorkFlow языков [Электронный ресурс] / А. Михеев, М. Орлов // PC Week/RE. 2005. № 36. Режим доступа: http://wf.runa.ru/rus/images/c/c1/Stat_ya4.pdf.

29. Месарович, М. Общая теория систем: математические основы / М. Месарович, Д. Михайло, Я. Такахара. М.: Мир, 1978. 311 с.

30. Репин, В.В. Процессный подход к управлению. Моделирование бизнес-процессов / В.В. Репин, В.Г. Елиферов. М.: РИА «Стандарты и качество», 2004. 408 с.

31. Сапегин, А. Реорганизация бизнес-процессов: как выглядит наше будущее? [Электронный ресурс] / А.Сапегин. Режим доступа: http://www.interfase.ru.

32. Сомерсвиль, И. Инженерия программного обеспечения / И.Сомерсвиль; пер. с англ. 6-е изд. М.: Вильямс, 2002. 624 с.

33. Спольски, Д. Лучшие примеры разработки ПО / Д. Спольски. СПб.: Питер, 2007. 208 с.

34. Туманов, В.Е. Проектирование хранилищ данных для приложений систем деловой осведомленности (Business Intelligence Systems) [Электронный ресурс] / В.Е. Туманов. Режим доступа: http://www.intuit.ru/department/database/bispowerd/14/4.html.

35. Деревянко, А.С. Технологии и средства консолидации информации: учеб. пособие / А.С. Деревянко, М.Н. Солощук. Харьков: НТУ «ХПИ», 2008. 432 с.

36. Орлов, С. Технологии разработки программного обеспечения: учебник / С. Орлов. СПб.: Питер, 2002. 464 с.: ил.

37. Технология моделирования бизнес-процессов [Электронный ресурс] / НИЦ CALS-технологий «Прикладная логистика». Режим доступа: http:// http://www.cals.ru.

38. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер. М.: Вильямс, 2004. 544 с.

39. Фаулер, М. UML в кратком изложении. Применение стандартного языка объектного моделирования / М. Фаулер, К. Скот. М.: Мир, 1999. 191 с.

40. Халл, Э. Разработка и управление требованиями (Практическое руководство пользователя) / Э.Халл. М., 2005.

41 Якобсон, А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо. СПб.: Питер, 2002. 496 с: ил.

42. IDEF [Электронный ресурс] / Режим доступа: http:// www.idef.com.

43. IDEF [Электронный ресурс] / Режим доступа: http://www.idef.com.

44. http://ru.wikipedia.org/wiki/XMI.

45. http://www.omg.org/technology/documents/bms_spec_catalog.htm.

46. http://www.pandia.ru/text/77/20/11377.php.

47. http://www.interface.ru/home.asp?artId=9067.

48. http://www.interface.ru/home.asp?artId=26579.

49. XML Schema – http://www.w3.org/XML/Schema/.

50. WSDL – http://www.w3.org/TR/wsdl/.

51. UML – http://www.omg.org/technology/documents/profile_catalog.htm.

52. BPMN – http://www.omg.org/technology/documents/bms_spec_catalog.htm.

53.UML спецификация. – www.omg.com.