

Э.А. Бабкин, О.Р. Козырев

**Архитектура  
и технология  
использования  
современных  
ERP-систем**

**Э.А. Бабкин, О.Р. Козырев**

**АРХИТЕКТУРА  
И ТЕХНОЛОГИЯ ИСПОЛЬЗОВАНИЯ  
СОВРЕМЕННЫХ ERP-СИСТЕМ**

---

**Учебное пособие**

# Оглавление

<b>Глава 1. Современные принципы использования информационных систем</b> .....	<b>11</b>
1.1. КИС — целостная платформа управления бизнесом .....	12
1.2. Ориентация на процессно-ориентированные методы управления .....	18
1.2.1. Организации, занимающиеся разработкой стандартов Workflow Management .....	26
1.2.2. Модель Workflow Management с точки зрения коалиции WfMC .....	28
1.3. Машино-ориентированные языки определения бизнес-процессов .....	33
1.3.1. История стандартов WfMC на языки определения Workflow .....	35
1.3.2. История стандартов и языки описания бизнес-процессов коалиции BPMI .....	36
1.3.3. История языка BPEL4WS .....	37
1.3.4. Другие стандарты .....	37
1.3.5. Тенденции развития стандартов на языки описания workflow .....	38
1.4. Неформальное введение в язык XPDL .....	40
1.4.1. Структура языка XPDL .....	42
1.4.2. Состав мета-модели языка XPDL .....	48
1.4.3. Описание атрибутов стандартных элементов языка XPDL .....	52
1.4.4. Пример описания workflow на языке XPDL .....	52
1.5. Определения типов данных, используемых в бизнес-процессе .....	58
1.6. Приложение к гл. 1. Математические основы автоматизированного управления бизнес-процессами .....	59

1.6.1. Теория сетей Петри .....	59
1.6.2. Ограниченность WF-языков, основанных на теории сетей Петри .....	60
1.6.3. Концепция Pi calculus .....	60
1.6.4. Понятия модели и мета-модели, их совместное использование в современных информационных технологиях .....	61

## **Глава 2. Редактор спецификации Workflow-процессов .....** 69

2.1. Редактор спецификаций JaWE (Java Workflow Editor) .....	70
2.2. Уровень пакета (Package Level) .....	73
2.3. Уровень процесса (Process level) .....	76
2.3.1. Функция общего вида (Generic Activity) .....	78
2.3.2. Текстовое представление (Text view) .....	80
2.3.3. XPDL-представление (XPDL View) .....	80
2.3.4. Проверка правильности XPDL-документа ....	81
2.3.5. Предопределенные комбинации клавиш в редакторе JaWE .....	82
2.4. Создание бизнес-процесса в среде разработки JAWE .....	83
2.4.1. Шаг 0. Инсталляция продукта .....	83
2.4.2. Шаг 1. Начало работы .....	84
2.4.3. Шаг 2. Создание процесса .....	85
2.4.4. Шаг 3. Запись в файл и создание пакета ....	88
2.4.5. Шаг 4. Работа над бизнес-процессом: создание функции «Получение заказа» (Receive Order) .....	90
2.4.6. Шаг 5. Работа над бизнес-процессом: определение данных процесса и пользова- тельских типов данных .....	92
2.4.7. Шаг 6. Работа над бизнес-процессом: определение приложения для исполнения функции «получение заказа» .....	99
2.4.8. Шаг 7. Работа над бизнес-процессом: определение подпроцесса «Проверка платежеспособности» (Check Credit) .....	103



2.4.9.	Шаг 8. Работа над бизнес-процессом: определение параметров вызова под- процесса «Проверка платежеспособности» из основного процесса .....	108
2.4.10.	Шаг 9. Работа над бизнес-процессом: создание функций «Аннулировать заказ» (Decline) и «Завершить заказ» (Finish_Order) .....	108
2.4.11.	Шаг 10. Работа над бизнес-процессом: определение правил перехода от функции «Проверка платежеспособности» к функции «Аннулировать заказ» или «Завершить заказ» .....	110
2.4.12.	Шаг 11. Работа над бизнес-процессом: уточнение всех деталей в основном процессе .....	112
2.5.	Контрольные задания к гл. 2 .....	114
2.6.	Приложение к гл. 2 .....	114

### **Глава 3.** Распорядитель выполнения работ ..... 126

3.1.	Общая информация о системе Shark .....	126
3.2.	Краткий обзор архитектуры Shark .....	128
3.3.	Пример настройки и работы Shark .....	131
3.3.1.	Инсталляция продукта .....	131
3.3.2.	Структура каталогов и основные программы администрирования.....	134
3.3.3.	Примеры работы с администраторским приложением Shark .....	134
3.4.	Контрольные задания к гл. 3 .....	161
3.5.	Приложения к гл. 3. Описание методов вызова внешних систем .....	162

### **Глава 4.** Архитектура и технология использования современных ERP-систем на примере системы OFBIZ ..... 166

4.1.	Введение .....	166
4.2.	Общая информация о системе OFBIZ .....	170

4.3.	Краткий обзор архитектуры OFBIZ .....	173
4.3.1.	Service Engine — центр управления службами (ЦУС) .....	177
4.3.2.	Entity Engine — центр управления элементами данных (ЦУЭД) .....	179
4.3.3.	Подсистема пользовательского Web-интерфейса .....	182
4.4.	Взаимодействие элементов системы OFBIZ .....	187
4.5.	Инсталляция системы .....	193
4.6.	Характеристика стандартных пользовательских приложений системы OFBIZ .....	197
4.7.	Контрольные задания к гл. 4 .....	199
<b>Глава 5.</b>	<b>Методы создания Web-интерфейсов в системе OFBIZ .....</b>	<b>201</b>
5.1.	Типы обработчиков .....	201
5.2.	OFBIZ-Widgets .....	201
5.2.1.	Базовые принципы. Структура файла .....	203
5.2.2.	Модель данных .....	207
5.2.3.	Содержимое разделов описания .....	212
5.3.	Стандартные составные элементы .....	219
5.3.1.	Формы .....	220
5.3.2.	Меню .....	228
5.3.3.	Деревья .....	231
5.3.4.	Использование шаблона проектирования «Декоратор» .....	234
5.4.	Методы изучения функциональности сложных программных систем .....	238
5.5.	Особенности интеграции с системой OFBIZ .....	244
5.6.	Готовый пример с ftl .....	247
5.7.	Контрольные задания к гл. 5 .....	250
5.8.	Приложения к гл. 5 .....	251
5.8.1.	Вопросы локализации интерфейса .....	251
5.8.2.	Информация по каскадным стилевым таблицам .....	254
5.8.3.	Внедрение .....	255
5.8.4.	Встраивание в теги документа .....	258

5.8.5. Импортирование .....	259
5.8.6. Связывание .....	259
5.8.7. Свойства элементов, управляемых с помощью CSS .....	260
Свойства цветов .....	261
Свойства текста .....	262

## **Глава 6. Создание Web-приложения в системе OFBIZ..... 263**

6.1. Постановка задачи. ....	263
6.1.1. Структура каталогов для компонента .....	266
6.1.2. Адаптация стандартных конфигураци- онных файлов компонента .....	266
6.2. Определение правил обработки запросов в Web-приложении .....	269
6.3. Разработка структуры Web-интерфейса .....	272
6.3.1. Описание структуры экранов на языке OFBIZ-Widget .....	272
6.3.2. Шаблоны системы FreeMarker .....	273
6.3.3. Логическая структура файла MainScreens.xml .....	273
6.4. Формы на языке OFBIZ-Widget .....	277
6.5. Назначение и приемы использования системы управления содержанием (Content Manager) в OFBIZ .....	279
6.5.1. Связь элементов содержания (Association) ...	284
6.5.2. MetaData .....	285
6.5.3. DataResource .....	285
6.6. Графические приложения для работы с содержимым .....	287
6.7. Использование возможностей CMS для реализа- ции информационной системы .....	292
6.7.1. Элементы языка OFBIZ-Widget и FTL-тэги для доступа к содержимому Content .....	292
6.7.2. Использование Survey .....	293
6.7.3. Применение CMS при создании интер- фейса системы biminfo .....	298

6.7.4. Загрузка данных в систему OFBIZ из внешних файлов .....	300
6.7.5. Конечный результат .....	301
6.8. Контрольные задания к гл. 6 .....	302
<b>Глава 7. Принципы использования и реализации Служб в системе OFBIZ .....</b>	<b>304</b>
7.1. Основное назначение и возможности Центра управления службами .....	304
7.2. Архитектура ЦУС .....	308
7.3. Правила определения модели служб .....	309
7.4. Правила вызова служб из Java-классов .....	313
7.5. Правила реализации служб на языке Java .....	315
7.5.1. Специфика вызова служб в ходе обра- ботки Web-запроса (controller.xml) .....	318
7.5.2. Инициализация служб на старте компонента .....	322
7.5.3. Интерактивный запуск служб из Web- приложения webtools .....	324
7.6. Создание службы для Web-приложения biminfo ...	325
7.7. Контрольное задание к гл. 7 .....	338
<b>Глава 8. Принципы использования центра управления элементами данных (Entity Engine) в системе OFBIZ .....</b>	<b>341</b>
8.1. Необходимые базовые сведения по современным принципам информационного моделирования и обработки данных .....	341
8.2. Архитектура ЦУЭД в системе OFBIZ. ....	359
8.3. Правила определения элементов данных и связей между ними (модель данных модуля OFBIZ) .....	363
8.3.1. Файл с определениями элементов данных (entitymodel.xml) .....	363
8.4. Примеры определения модели .....	370

8.5. Инициализация описаний элементов данных на старте модуля .....	374
8.6. Способы манипулирования элементами данных в Java-классах .....	376
8.7. Примеры кода .....	381
8.8. Пример использования ЦУЭД в Web-приложения biminfo .....	386
8.9. Контрольные задания к гл. 8 .....	393
8.10. Приложение к гл. 8 .....	393
<b>Глава 9. Автоматизации бизнес-процессов в системе OFBIZ. Принципы разработки приложений .....</b>	<b>397</b>
9.1. Автоматизация бизнес-процессов с помощью OFBIZ Workflow Engine .....	397
9.2. Оригинальный OFBIZ Workflow Engine .....	398
9.2.1. Реализованные типы функций в описании бизнес-процессов .....	398
9.2.2. Особые расширения в XPDЛ-описании .....	399
9.2.3. Использование workflow Engine .....	399
9.3. Возможности проинтегрированного Workflow Engine Shark в составе системы OFBIZ .....	401
9.3.1. Расширенные атрибуты в XPDЛ-определении бизнес-процесса, используемые в Shark Workflow Engine .....	406
<b>Глава 10. Программный проект по разработке информационной системы .....</b>	<b>408</b>
10.1. Постановка задачи .....	408
10.1.1. Преамбула .....	408
10.1.2. Задание к разработке .....	409
Общее описание задачи .....	409
Список требований .....	409
10.1.3. Организация разработки и роли участников .....	411
10.2. Общие рекомендации по принципам разработки в системе OFBIZ .....	413
<b>Библиографический список .....</b>	<b>414</b>

## Предисловие

Данное учебное пособие составлено в соответствии с программой курса «Архитектура и технология использования корпоративных информационных систем», который читается авторами студентам направлений подготовки 080700 «Бизнес-информатика» и 010500 «Прикладная математика и информатика» Нижегородского филиала Государственного университета — Высшая школа экономики.

В данной книге авторы рассматривают ряд вопросов, возникающих перед специалистами, занятыми вопросами разработки и использования корпоративных информационных систем (ERP). Конкретные детали, описание проблем, возникающих в ходе настройки и внедрения КИС, разбираются на примере широко известной системы с открытым кодом OFBIZ.

Пособие состоит из десяти глав, в которых приведен достаточный фактический материал. Пояснение теоретического материала подкреплено набором рассмотренных в тексте примеров. В конце каждой главы помещен ряд справочных материалов, а также набор несложных заданий. Их самостоятельное решение настоятельно рекомендуется авторами для более полного усвоения рассматриваемого материала.

Курс составлен под влиянием классических курсов по ERP-системам, а также исходя из личного опыта авторов в качестве разработчиков таких систем. Все пожелания авторов будут приняты с благодарностью. Книга подготовлена и издается в рамках Инновационной образовательной программы ГУ-ВШЭ «Формирование системы аналитических компетенций для инноваций в бизнесе и государственном управлении».

# Глава 1 \_\_\_\_\_

## Современные принципы использования информационных систем

Повсеместное использование компьютеров на предприятиях для повышения эффективности процедур накопления, хранения и преобразования информации стало значительным явлением в современном обществе. При этом автоматизация отдельных производственных или управленческих функций, как-то бухгалтерский учет или сбыт готовой продукции, считается уже пройденным этапом для многих организаций, и качественное изменение в эффективности использования информационных технологий возможно только в том случае, когда различные программные системы интегрированы в единое решение — *корпоративную информационную систему* (КИС), объединяющую на основе современных информационных технологий все функциональные подсистемы различных уровней управления [9, 10, 11].

*Корпоративность* означает соответствие информационной системы нуждам крупной фирмы, имеющей сложную территориальную структуру. Информационная система отдельных подразделений фирмы (финансовых, экономических, маркетинговых и других) не может претендовать на корпоративность. Только полнофункциональная система может по праву быть охарактеризована как КИС.

КИС основана на продуманных концептуальных решениях, а также обязательно должна иметь в своем составе развитое математическое, программное, сетевое, техническое (компьютерное) и организационное обеспечение.

*Концептуальные решения* определяют цели, задачи, степень автоматизации и стратегии информационной системы,

этапы и перспективы реализации системы, необходимую реконфигурацию организационной структуры, иерархию уровней принятия решений: реперные технические и программные средства, источники финансирования, кадровую политику.

*Математическое обеспечение (МО)* позволяет подготовить множество альтернатив решений в соответствии с целями, задачами, стратегией и постоянными или временными приоритетами и граничными условиями. Отсутствие МО однозначно доказывает, что некоторая система, преподносимая в качестве КИС, в сущности таковой не является.

*Программное обеспечение (ПО)* реализует требования, заложенные математическим обеспечением, а также автоматизирует операции сбора (ввода), обработки, хранения, движения, формирования и вывода информации.

*Сетевое обеспечение* является техническим средством организации высокоскоростного управляемого обмена информацией между территориально распределенными подсистемами КИС.

*Техническое (компьютерное) обеспечение* включает все средства ввода, хранения, обработки и вывода информации.

*Организационное обеспечение* подразумевает не только наличие штата, обслуживающего все составляющие КИС, но и обоснованную оптимизацию всей организационной структуры предприятия для наиболее эффективного использования возможностей системы.

## ***1.1. КИС — целостная платформа управления бизнесом***

По сути своего определения КИС является системой управления предприятием в целом, ее основная задача — поддержка его функционирования и развития. Смысл существования любого коммерческого предприятия, как известно, — получение прибыли. Соответственно результаты создания и эксплуатации КИС должны позволять предприятию быстро увеличивать прибыль, прежде всего за счет повышения эффективности управления на различных уровнях.



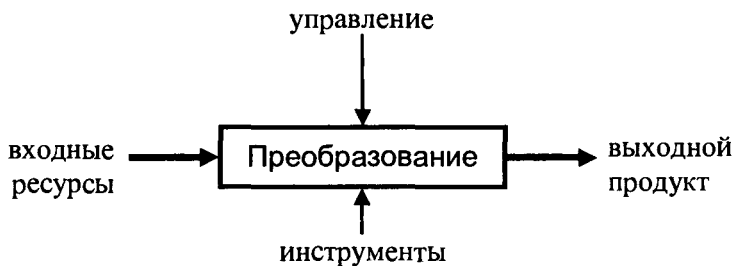


Рис. 1.1. Схема деятельности предприятия

Несмотря на то, что сферы деятельности предприятий (производство, услуги) могут быть самыми разными, в общем виде задачи управления схожи. Они заключаются в наилучшей организации преобразования поступающих на вход ресурсов в конечный результат, к которому проявляют интерес потенциальные клиенты. В процессе преобразования предприятие использует определенные инструменты и управляющие воздействия (рис. 1.1).

Таким образом, структура и принципы функционирования любой организации могут быть описаны характерными объективными законами управления, регламентирую-

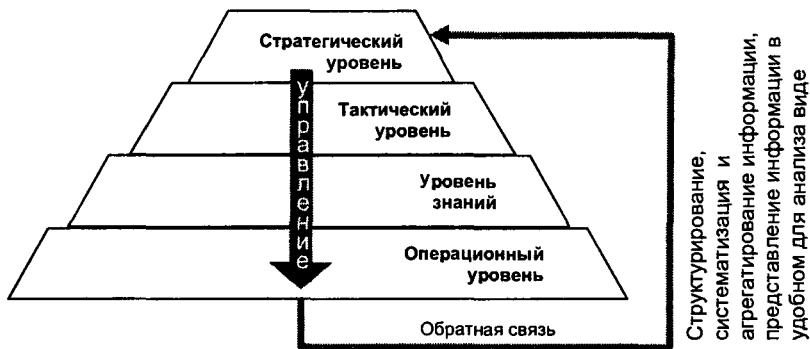


Рис. 1.2. Организация замкнутого цикла управления в КИС

щими управляющие воздействия на систему. Эти законы управления находят свое воплощение в свойствах и структуре использующейся на предприятии информационной системы.

С такой общей точки зрения структура КИС образует замкнутый цикл управления, связывающий в единую структуру управления и контроля различные уровни управления (рис. 1.2).

Исходными данными для КИС служат данные о главных ресурсах (финансовых, материальных, кадровых, информационных), которыми необходимо управлять и которые на выходе трансформируются в результат основной деятельности предприятия. Эти данные собираются и накапливаются на операционном уровне. По мере их движения вверх по управленческой пирамиде происходит структурирование первичной информации, ее отбор, и отчеты для высшего руководства содержат несколько значимых для выработки стратегических величин.

На тактическом уровне и уровне знаний КИС хранит и обрабатывает структурированные (т.е. систематизированные в соответствии с требованиями среднего управляющего персонала) корпоративные данные. На этих уровнях КИС используется менеджерами среднего звена для получения информации, интересующей конкретно их.

Стратегический уровень содержит системы поддержки принятия решений, которые могут включать ситуационные центры, средства многомерного анализа данных и прочие инструменты аналитической обработки. Используемые здесь специальные математические методы позволяют прогнозировать динамику различных показателей, анализировать затраты по разным видам деятельности, уяснять их детальную структуру, формировать подробные бюджеты по разным схемам.

Такая структура КИС позволяет говорить о создании на предприятии системы управления в кибернетическом<sup>1</sup> смысле этого понятия. Действительно, в такой системе существуют все составляющие части кибернетической системы управления: субъект управления (тот, кто управляет), объект управления (то, чем управляют), канал передачи

---

<sup>1</sup> См. понятие «кибернетика» в толковом словаре.

управляющего воздействия, информационный канал обратной связи (рис. 1.3).

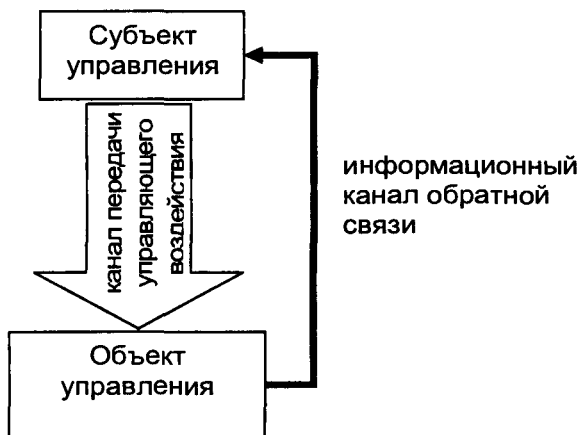


Рис. 1.3. Принципиальная структура кибернетической системы

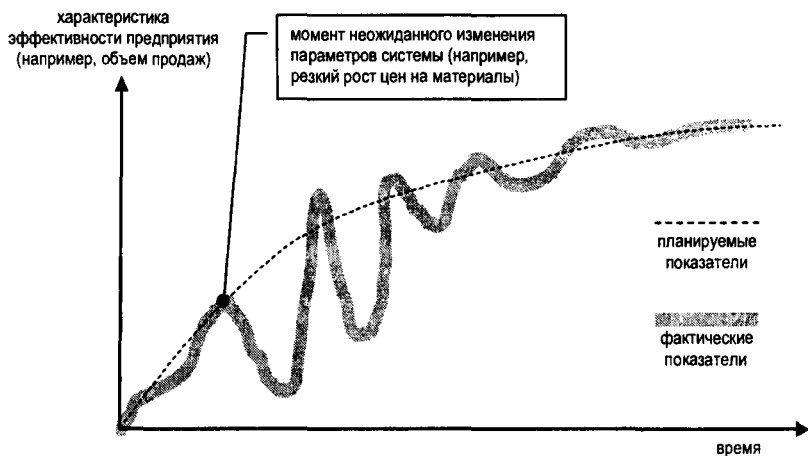


Рис. 1.4. Иллюстрация устойчивости кибернетической системы к воздействию возбуждающих факторов

Появление такой системы способствует более эффективному управлению предприятием как сложной системой, позволяя гасить влияние внешних или внутренних мешающих воздействий (так называемых возбуждающих факторов). Система становится устойчивой к таким воздействиям и стремится сохранять свое заданное состояние (рис. 1.4).

Такой системный подход к определению целей КИС в процессе управления заставляет критически пересмотреть основные принципы разработки и применения информационных систем на предприятии. При переходе от использования информационных систем первого поколения, ограниченных решением изолированных задач нескольких подразделений, к применению единой корпоративной системы главнейшая роль отводится разработке согласованной управленческой методологии. Такая методология должна объединять бизнес-стратегию предприятия и передовые информационные технологии. Причем на первом месте находится отработанная структура управления предприятием, а программные и аппаратные средства автоматизации служат для выполнения второстепенных, инструментальных задач.

В большинстве современных управленческих методологий используется многолетний международный опыт, выраженный в форме стандартов управления бизнесом. К числу наиболее известных стандартов, без всякого сомнения, можно отнести разработку американской исследовательской компании Gartner Group под названием «Планирование ресурсов в масштабе предприятия» (в оригинале — Enterprise Resource Planning (ERP)) [9, 11].

В основе ERP лежит принцип создания единого хранилища данных, содержащего всю деловую информацию, накопленную организацией в процессе ведения деловых операций, включая финансовую информацию, данные, связанные с производством, управлением персоналом, или любые другие сведения. Это устраняет необходимость в передаче данных от системы к системе. Кроме того, любая часть информации, которой располагает данная организация, становится одновременно доступной для всех работников, обладающих соответствующими полномочиями. Таким образом, методы ERP претендуют на управление всеми

ресурсами, имеющимися у предприятия (персоналом, финансами). При этом ERP лучше учитывает корпоративную структуру предприятия, его масштаб, что позволяет достичь конкурентных преимуществ за счет оптимизации бизнес-процессов предприятия и снижения издержек.

Методологии управления на основе стандарта ERP получили широкое распространение, прежде всего в производственном секторе, поскольку планирование всех ресурсов позволило сократить время выпуска продукции, снизить уровень товарно-материальных запасов, а также улучшить обратную связь с потребителем при одновременном сокращении административного аппарата. На практике концепция управления ERP может быть реализована либо одной интегрированной КИС, либо набором разнородного программного обеспечения. В последнем случае один из программных продуктов используется как базовый, а другие специализированные программные продукты интегрируются как бы «поверх» базового. В любом случае для того, чтобы соответствовать стандарту ERP, современная КИС должна содержать следующие подсистемы:

- ◆ управление цепочкой поставок (Supply Chain Management, SCM, ранее — Distribution Resource Planning, DRP);
- ◆ усовершенствованное планирование и составление расписаний (Advanced Planning and Scheduling, APS);
- ◆ модуль автоматизации продаж (Sales Force Automation, SFA);
- ◆ автономный модуль, отвечающий за конфигурирование (Stand Alone Configuration Engine, SCE);
- ◆ окончательное планирование ресурсов (Finite Resource Planning, FRP);
- ◆ интеллектуальный анализ бизнеса, OLAP-технологии (Business Intelligence, BI);
- ◆ модуль электронной коммерции (Electronic Commerce, EC);
- ◆ управление данными об изделии (Product Data Management, PDM).

Отметим, что информационные системы, соответствующие стандарту ERP, часто называют просто ERP-системами. Такие системы реализуют контроль над всеми потоками

данных, способствуют эффективному управлению удаленными филиалами и сбытовыми подразделениями по всей территории действия крупной компании, а также развитию электронного бизнеса. Системы на основе этого стандарта созданы как раз для управления себестоимостью продукции и достижения за счет этого конкурентных выгод и реализуют методы планирования и управления, позволяющие:

- ◆ регулировать количество запасов, устраняя их дефицит и залеживание, и тем самым значительно снизить омертвленные в запасах затраты и складские издержки;

- ◆ сократить незавершенное производство, поскольку производство планируется только на основе спроса на конечную продукцию, при этом производственные работы инициируются исходя из срока, к которому должен быть выполнен клиентский заказ;

- ◆ оценивать выполнимость поступивших заказов с точки зрения имеющихся на предприятии мощностей;

- ◆ сократить расходы и время, затрачиваемые на изготовление продукции, за счет оптимизации бизнес-процессов;

- ◆ отслеживать фактическую производительность каждой производственной единицы и, сравнивая ее с плановой производительностью, оперативно вносить корректировки в производственные планы;

- ◆ в результате уменьшения цикла производства и цикла выполнения заказа более гибко реагировать на спрос;

- ◆ улучшить обслуживание клиентов и заказчиков за счет своевременного исполнения поставок.

## ***1.2. Ориентация на процессно-ориентированные методы управления***

Использование единой методологии управления всеми подсистемами предприятия на основе информационных технологий заставляет искать новые теоретические принципы взаимодействия различных подразделений, отказываясь от привычного для многих организаций функционального подхода к управлению. Напомним, что в этом случае все работы разбиваются на отдельные задачи и распределяются между отдельными специалистами, каждый из которых отвечает только за свою операцию. Такой подход сыграл положительную роль в начале двадцатого века, позволив использовать

персонала с низкой квалификацией для конвейерной сборки сложных изделий массовыми партиями.

Но в настоящее время все окружающее нас меняется с огромной скоростью. С развитием современных технологий и с ростом конкуренции производство утратило стабильность. Изменился и потребитель — он хочет иметь выбор и выбирает лучшее. В результате производитель вынужден непрерывно осваивать новые технологии для того, чтобы соответствовать постоянно растущим запросам своих клиентов. Понятия «развитие» и «изменение» составляют саму сущность любой сегодняшней экономической деятельности. Поэтому забота об эффективности организации производства и ее постоянном совершенствовании становится повседневной задачей как для руководства, так и для рядовых сотрудников. В такой ситуации функциональный подход является серьезным препятствием для внесения оперативных изменений, поскольку практически не решается задача мотивации работника, его заинтересованности в общем деле. Действительно, в функционально-ориентированной организации работник лишь механически выполняет функции, которые ему поручает руководитель, по технологиям, определенным тем же руководителем. Работник не знает ни общих целей того предприятия, где он трудится, ни своего места в общем деле, ни того, как его работа сказывается на процессе в целом и на конечном результате процесса. Роль руководителя заключается в том, чтобы предписывать работнику, КАК выполнять его ЗАДАЧУ, а если технология процесса изменяется, то вносить свои коррективы и устранять появившиеся производственные проблемы «сверху».

Очевидно, что должны быть изменены основополагающие принципы распределения обязанностей между работниками и методы контроля за выполнением задач. Эти принципы должны учитывать, что деятельность того или иного работника или подразделения важна не сама по себе, а в контексте достижения стратегических целей на уровне предприятия. Роли, обязанности и используемые в производстве продукта материалы образуют взаимозависимую систему. Поэтому в ходе автоматизации необходимо учитывать связи между различными подразделениями.

Основным понятием, позволяющим явно определять роли и взаимосвязи различных участников решения единой задачи (производства продукта), является понятие процесса (также используются термины «трудовой процесс» или «бизнес-процесс», от англ. business process) [9, 12, 13].

Можно привести несколько формулировок этого понятия, отражающих различные характеристики. Например, ГОСТ Р ИСО 9000—2001 дает такое определение:

*«Процесс — совокупность взаимосвязанных и взаимодействующих видов деятельности, преобразующая входы в выходы».*

Следующее, более прагматичное определение подчеркивает, что процесс должен управляться (корректироваться) и его надо строить оптимально:

*«Процесс — это действие, направленное на достижение результата, которое может корректироваться при его выполнении и которое сосредоточено на достижении конечного результата при оптимальном использовании ресурсов».*

Наконец, можно дать очень подробное определение, которое может служить основой для построения КИС, которая будет ориентирована именно на «сквозную»<sup>2</sup> автоматизацию бизнес-процессов, связывая в единую цепочку по получению конечного продукта различные подразделения, сотрудников, программные и технические средства:

*«Процесс (process) — упорядоченная последовательность отдельных «задач» (task). Каждая задача имеет «входы» (Inputs) и «результаты» (Outputs) — некоторые наборы объектов реального мира. В качестве объекта может выступать все что угодно: оборудование, выдача принтера (на листе бумаги), информация в электронном виде и т.д.*

*Каждая задача выполняется некоторым исполнителем. Для обозначения исполнителя вводится термин «роль» (role). Ролью может быть либо человек, либо некоторое устройство или программа.*

*Исполнитель функционирует в соответствии с определенными «правилами» (rules). Степень формальности правил может быть самой разной. Часто процесс распадается на задачи, исполняемые в различных организационных подразделениях (а может оказаться — и на различных территориях). Для контроля за правильностью исполнения процесса вводится*

<sup>2</sup> См. определение «сквозная» автоматизация в толковом словаре.



*ответственный за процесс — «хозяин процесса» (owner). Это тоже роль».*

С использованием понятия бизнес-процесса практически вся деятельность предприятия может быть определена формально. Такое описание обычно называют бизнес-моделью (Enterprise Model):

*«Бизнес-модель — это описание предприятия, как сложной системы, с заданной точностью. В рамках бизнес-модели отображаются все объекты (сущности), процессы, правила выполнения операций, существующая стратегия развития, а также критерии оценки эффективности функционирования системы. Форма представления бизнес-модели и уровень ее детализации определяются целями моделирования и принятой точкой зрения».*

Бизнес-модель позволяет создать целостную картину взаимодействия различных бизнес-процессов, которая не ограничивается рамками отдельных подразделений.

Процесс создания и анализа бизнес-моделей получил название Моделирование Предприятия или Бизнес-Моделирование (Enterprise Modeling). Бизнес-моделирование — это многогранная концепция, которая впервые была использована в авиационной промышленности США в конце 80-х гг.

Бизнес-модель позволяет создать целостную картину взаимодействия бизнес-процессов, которая не ограничивается рамками отдельных подразделений. В случае целенаправленного подхода к управлению на основе процессов появляется реальная возможность создать такую атмосферу на предприятии, в которой каждый работник знает, ЧТО и ДЛЯ ЧЕГО он должен делать. Право принятия решений и ответственность за них передается работникам, компетентным в своих областях. Работник совершенно четко и определенно знает и цели своего предприятия, и свою роль в общем деле. При необходимости он может сам адаптировать процесс достижения результата к новым условиям или требованиям. Здесь важно, что каждый осознанно и творчески занимается своим трудом, понимая сущность происходящих на предприятии процессов, и может активно участвовать в их изменении. Критерием оценки любой работы является эффективность целостного процесса.

Процессное управление на сегодня — ведущий, лидирующий метод управления бизнесом. Резкий подъем экономики Японии произошел именно тогда, когда японские компании перешли на процессное управление. Более того, в настоящее время компании ценятся не только производимыми продуктами, но и существующими у них бизнес-процессами. Бизнес-процессы патентуются, покупаются, и на базе уже запатентованных бизнес-процессов строятся новые высокоэффективные компании. Возможно, это пока не характерно для нас, но на Западе — это бизнес, объект купли-продажи. Запатентованные бизнес-процессы аналогичны ноу-хау. Динамический характер бизнес-процессов и явная фиксация большинства ролей и обязанностей позволяют активно реагировать на внешние и внутренние изменения, гибко перестраивая организационную структуру, процесс производства и даже специализацию компании. Такая деятельность получила название «бизнес-реинжиниринг».

В процессе «бизнес-реинжиниринга» при переходе от функционального к процессному принципу управления основная задача заключается в том, чтобы с использованием средств моделирования явно описать бизнес-процессы и сделать их основой реализации управления организацией. Как только бизнес-процессы будут явно выражены, описаны и объединены в единую формальную бизнес-модель, появится возможность исследовать описанные процессы, целенаправленно управлять ими, а также проводить их сквозную автоматизацию.

Примеры бизнес-процессов, которые можно последовательно прописать, формализовать и управлять, достаточно разнообразны — это обработка заявок, сервисное обслуживание клиентов (телекоммуникационные компании, страховые компании, медицинские компании), выдача патента, оказание технической поддержки, подготовка и согласование контрактов, открытие счета, выдача ссуды в банке и т.д.

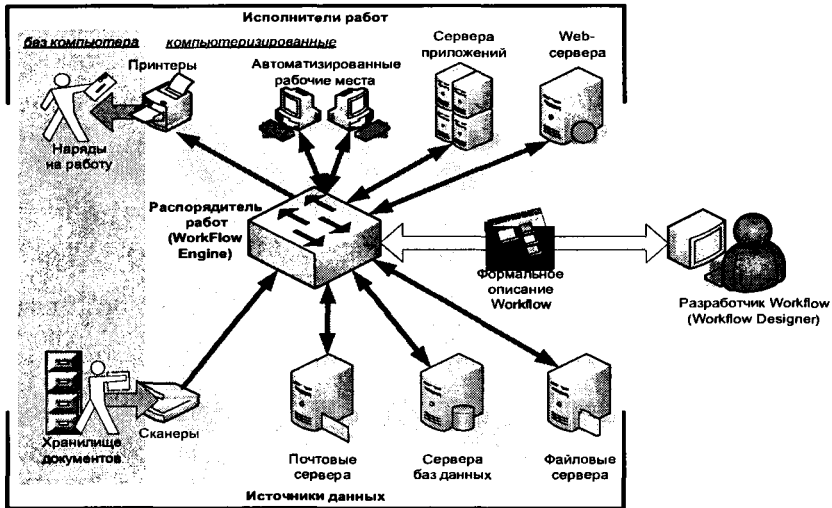
На сегодня наиболее удачной информационной технологией для формализации и автоматизации выполнения таких бизнес-процессов является технология Workflow Management. Термин Workflow дословно переводится как «поток работ». Поток работ рассматривается в рамках

бизнес-процесса. Таким образом, Workflow Management как технология — это автоматизированное управление потоком работ и через него бизнес-процессом. Согласно статистике, в функционально ориентированных бизнес-системах в среднем 20% времени тратится на выполнение функции (операции), а 80% на поиск информации и передачу результатов выполнения функции (операции). Технология Workflow Management призвана автоматизировать прием и передачу информации от одного рабочего места другому. Таким образом, сотруднику на своем рабочем месте нет необходимости задумываться, откуда пришло задание и куда его передать. Именно на этапе передачи возникает большое количество ошибок, которое вносит свою лепту в 80% потери времени. Workflow Management предотвращает эти ошибки. Важно отметить, что Workflow Management автоматизирует процессы, а не отдельные функции. Системы, построенные по технологии Workflow Management, предоставляют средства для автоматического отслеживания последовательности и времени выполнения функций, маршрута документов, контроля загрузки участников процесса на различных его стадиях.

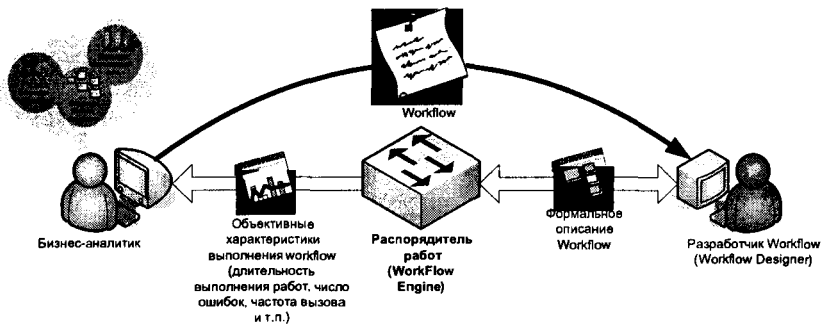
Автоматизация функции связана с работой с базами данных и банками информации. На рабочем месте сотрудник вызывает информацию, обрабатывает и отправляет результат в базы данных или банки информации. Автоматизация процесса строится на уже описанном средствами моделирования процессе. Принципиально важными являются последовательность выполнения заданий и конкретные исполнители, становится ясно, какая информация необходима сотрудникам для выполнения функции и куда, в каком виде должны отправиться результаты выполненного задания. Автоматизация процесса — путь радикального повышения производительности и качества основных производственных функций в рамках бизнес-процесса. Поэтому многие теоретики и практики рассматривают технологию Workflow Management как обязательный элемент эволюционного пути перехода от функциональной к процессной ориентации организации. Важной чертой технологии Workflow Management является интеграция (легкое встраивание) приложений (конкретных, традиционных про-

грамм используемых в организации), а также ручных, неавтоматизированных операций (рис. 1.5). Если сходу внедрять в организации систему класса ERP, зачастую приходится отказываться от всех приложений, что неизбежно влечет ряд проблем. Тратится большое количество времени на переобучение сотрудников работе с новыми приложениями, как следствие, растут сроки внедрения системы ERP. От 20 до 40% сотрудников, работавших на старых приложениях, не готовы к переходу на новые, вследствие чего люди уходят, и организация теряет квалифицированные кадры и знания. Применение технологии Workflow Management является промежуточным вариантом автоматизации. С одной стороны, мы работаем по-новому, т.е. в процессе, с другой стороны, работаем в старых приложениях. Организации, имеющие большое количество приложений, не собираются от них уходить, так как это требует значительных финансовых затрат, а покупать систему, в которой будут использоваться только 10% функций, невыгодно. Поэтому системы Workflow Management в этом плане значительно дешевле, чем системы класса ERP.

Понятно, что анализ ситуации должен быть быстрым, решения взвешенными, направленными на выполнение целевой функции. Очевиден оперативный контроль над протеканием всего процесса. В каждый момент времени мы должны располагать доступом к информации о том, в какой стадии находится процесс, что выполняется, кто ответственен за выполнение той или иной функции, обеспечивается аудит — кто, что сделал. В любой момент мы должны располагать возможностью получения отчета в любом интересующем нас разрезе. Такая возможность появляется в случае использования системы Workflow Management, потому что она объективно протоколирует все детали выполнения операций в бизнес-процессе (рис. 1.6). Бизнес-аналитик, используя эту информацию, а также свои знания об организационной структуре предприятия, вычислительных средствах и бизнес-модели, может оперативно выполнить необходимые коррективы в workflow. Это дает возможность постоянно корректировать правила выполнения бизнес-процессов в зависимости от изменения внутренних и внешних условий.



**Рис. 1.5.** Архитектура системы Workflow Management позволяет легко интегрировать различные методы выполнения операций и хранения данных в единый и динамичный бизнес-процесс



**Рис. 1.6.** Организация эффективной обратной связи в процессе управления с помощью системы Workflow Management

### *1.2.1. Организации, занимающиеся разработкой стандартов Workflow Management*

На сегодняшний день основными организациями, устанавливающими стандарты в области Workflow можно назвать:

- ◆ Workflow Management Coalition (WfMC);
- ◆ Business Process Management Initiative (BMPI.org);
- ◆ Workflow And Reengineering International Association (WARIA).

Коалиция WfMC (Workflow Management Coalition) является некоммерческой организацией, которая ставит своей целью расширение возможностей использования технологий Workflow Management путем разработки единой терминологии и стандартов. Коалиция WfMC основана в 1993 г. и сегодня включает в себя более 300 компаний, специализирующихся на разработке аппаратных и программных систем, внедрении, консалтинге, а также учебные заведения.

Исследования показали, что все продукты класса Workflow Management обладают некоторыми общими характеристиками, потенциально позволяющими им достичь определенного уровня взаимодействия за счет использования единых стандартов для различных функций. Коалиция WfMC была создана с целью идентификации этих функциональных областей и разработки соответствующих спецификаций для реализации в различных продуктах. Такие спецификации позволят обеспечить взаимодействие между разнородными продуктами Workflow Management и улучшить интеграцию этих продуктов с другими системами (например, электронная почта, системы распознавания, управление документами и т.д.), расширив тем самым возможности для эффективного использования технологии Workflow Management на рынке ИТ как в интересах пользователей, так и в интересах поставщиков этих технологий. Создание и успешная работа коалиции WfMC не случайны — организации, инвестирующие в проекты по технологии Workflow Management, хотя бы уверены, что их инвестиции достаточно хорошо защищены. Следование стандартам — это один из методов такой защиты, тем более, когда системы класса Workflow Management являются свя-

зующим звеном между различными системами в рамках организации.

За время своей работы WfMC в рамках рабочих групп были разработаны:

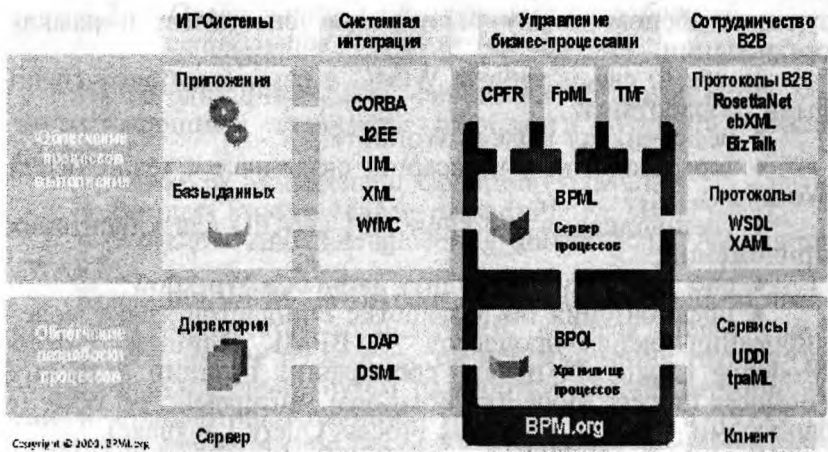
- ◆ референтная модель Workflow;
- ◆ терминология и глоссарий, связанные с технологией Workflow;
- ◆ спецификация Workflow API (WAPI) для клиентских приложений;
- ◆ спецификация по аудиту данных в Workflow;
- ◆ предпочтения по разработке стандартов языка моделирования бизнес-процессов — BPML (Business Process Modeling Language), правил составления нотаций — BPMN (Business Process Modeling Notations), интерфейс по обмену запросами — BPQL (Business Process Query Language).

Географически структура WfMC делится на регионы, каждый регион может состоять из одной или нескольких стран. Россия считается отдельным регионом. В настоящее время интересы России в WfMC представляет компания «Логика Бизнеса».

Группа BPMI.org (Business Process Management Initiative) — основана 7 августа 2000 г. как независимая некоммерческая организация. Группа BPMI.org ставит своей целью разработку и продвижение инициатив в области управления бизнес-процессами (Business Process Management, BPM) путем создания стандартов в области проектирования процессов, их автоматизации и оптимизации. Более 80 компаний, производителей BPM — модулей и систем, таких как IDS Scheer, Fujitsu, IBM, Hewlett-Packard, SAP, Sun, являются членами этой организации.

Группа BPMI.org вырабатывает стандарты и разрабатывает средства для решения задач интеграции бизнес-процессов. Основные документы, разработанные группой:

- ◆ BPML (Business Process Modeling Language) — метаязык для моделирования бизнес-процессов. BPML описывает абстрактную исполняемую модель бизнес-процессов компании. BPML представляет бизнес-процессы в виде совмещения потока управления, потока данных и потока событий с учетом бизнес-правил, ролей и правил безопасности. Благодаря этому языку осуществляется возможность



**Рис. 1.7.** Место стандартов BPMI в общем контексте технологии Workflow Management

обмена моделями процессов между различными системами управления бизнес-процессов;

- ♦ **BPMN** — нотации по моделированию бизнес-процессов (The Business Process Modeling Notation). Нотация BPMN позволяет в графическом виде представить бизнес-процессы как диаграмму бизнес-процессов (Business Process Diagram, BPD). BPMN-спецификация позволяет связываться между элементами графической нотации и структурированными языками, например BPML;

- ♦ **BPQL** — язык запросов бизнес-процесса (Business Process Query Language). Язык BPQL определяет стандартный интерфейс запросов между системами управления бизнес-процессами. Это позволяет системным администраторам и аналитикам анализировать и управлять системами УБП.

Технологические связи между языками, протоколами, сервисами иллюстрирует схема, изображенная на рис. 1.7.

### 1.2.2. Модель Workflow Management с точки зрения коалиции WfMC

В официальной глоссарии Workflow Management Coalition термин «поток работ» (Workflow) определяется как полная или частичная автоматизация бизнес-процесса, при





**Рис. 1.8. Взаимосвязь понятий технологии Workflow Management**

которой документы, информация или задания передаются для выполнения необходимых действий от одного участника к другому в соответствии с набором процедурных правил. Система управления Workflow Management описывает, создает, управляет потоком работ при помощи программного обеспечения, которое способно интерпретировать описание процесса, взаимодействовать с участниками потока работ и при необходимости вызывать соответствующие приложения. Приведем соотношение между основными понятиями, существующими при процессном управлении (рис. 1.8).

На перечисленные понятия опирается система Workflow Management. Мы описываем бизнес-процессы, а затем управляем ими. Описываемый бизнес-процесс может состоять из отдельных подпроцессов. В описание могут включаться отдельные функции, выполняемые как вручную, так и автоматически. Система Workflow Management управляет

всеми автоматизированными функциями, поддерживает процессы управления. Существуют отдельные экземпляры процессов, т.е. версии процессов, отличающиеся друг от друга потоками данных. Заявки нового клиента – очередная версия процесса, очередной его экземпляр. В результате мы получаем либо рабочие объекты, либо вызываем приложения. Если задача заключается в считывании информации из баз данных или в помещении информации, то мы вызываем эти приложения и производим с ними соответствующие действия. Основные компоненты системы Workflow Management изображены на рис. 1.9.

Описание процесса состоит из набора функций, а экземпляр процесса (конкретная версия процесса) – частный случай каждого процесса, с конкретными входными данными. Таким образом, рабочие задания относятся к каждому экземпляру процесса, в зависимости от разных условий (от того, как они протекают).

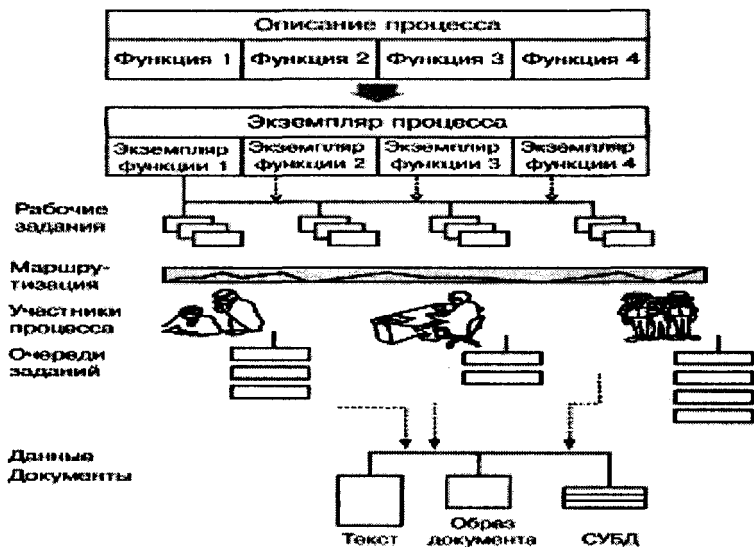


Рис. 1.9. Основные элементы технологии Workflow Management

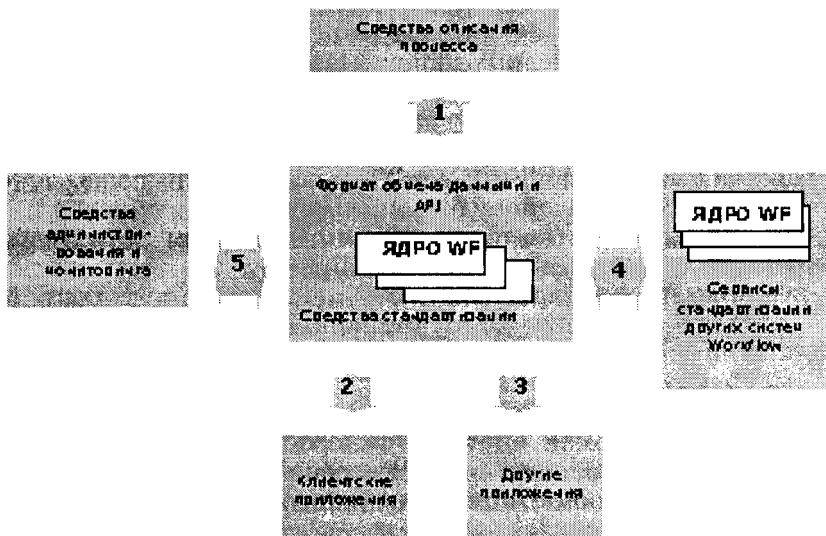


Рис. 1.10. Эталонная модель системы Workflow Management

**Маршрутизация.** В Workflow хранятся все маршруты процессов и конкретные версии процессов, поскольку версия может выполняться длительное время, и при этом возможны изменения. Может измениться сам процесс, а в исполнении находится предыдущая версия. Workflow это поддерживает. Участники процесса, т.е. исполнители, работающие на конкретных местах, фигурируют в процессе. Очереди заданий — система автоматически подает задания на рабочие места. Возможны приоритеты заданий, группы заданий или выполнение задания группой. Например, нам совершенно безразлично, кто из юристов будет рассматривать наш договор. Данные и документы — текст, образ документа, системы управления базами данных и т.п.

Коалицией WfMC разработано архитектурное представление системы управления Workflow, идентифицирующее важнейшие системные интерфейсы. Эталонная модель

представляет собой общую архитектуру взаимодействий, в рамках которой WfMC разрабатывает стандарты.

Эталонная модель Workflow Management Coalition представлена на рис. 1.10.

Она идентифицирует интерфейсы, охватывающие, в широком плане, пять функциональных областей взаимодействия между системой Workflow и ее средой:

- 1) импорт и экспорт описаний процессов;
- 2) взаимодействие с клиентскими приложениями и программой-обработчиком списка работ;
- 3) вызов программных инструментов или приложений;
- 4) взаимодействие между различными системами Workflow;
- 5) функции администрирования и мониторинга.

*Интерфейс №1.* Интерфейс ядра системы Workflow Management с инструментарием описания процессов. Он обеспечивает обмен описаниями процессов между инструментальными средствами BPR, репозиториями описаний процессов и ядром системы Workflow Management, составляющим домен обслуживания потоков работ.

*Интерфейс №2.* Интерфейс ядра с приложениями управления потоками работ. Он предназначен для интеграции клиентских приложений в системе, обеспечивающей переносимость и повторное использование приложений в различных workflow-средах.

*Интерфейс №3.* Интерфейс ядра с агентами, функционирующими в запущенных приложениях. Он предусматривает интеграцию программных агентов, которые открывают доступ к другим услугам среды, имеющим собственные стандартные API.

*Интерфейс №4.* Интерфейс ядра системы с другими доменами обслуживания потоков работ. Он обеспечивает функциональную совместимость процессов в нескольких workflow-системах, если один и тот же бизнес-процесс реализуется несколькими системами.

*Интерфейс №5.* Интерфейс ядра системы с инструментарием администрирования и мониторинга. Он содержит спецификации стандартных событий аудита и формата их записи, чтобы обеспечить интеграцию контрольных журналов, которые ведутся в различных workflow-системах.

### 1.3. Машино-ориентированные языки определения бизнес-процессов

В предыдущих разделах, рассматривая вопросы проектирования корпоративных информационных систем, мы обратили особенное внимание на технологию Workflow Management и на ее конкретное воплощение в виде автоматизированных систем Workflow Management по стандарту WfMC, рассматривали их достоинства, недостатки и область применения. Согласно концепции Workflow Management деятельность любого предприятия можно представить в виде набора бизнес-процессов. Бизнес-процесс — это упорядоченный по времени набор заданий, выполняемых как людьми, так и информационными системами, который направлен на достижение заранее установленной бизнес-цели к определенному сроку (рис. 1.11). Задача системы Workflow Management — сквозная автоматизация бизнес-процессов предприятия и контроль за их выполнением. Система должна выполнять две основные функции:

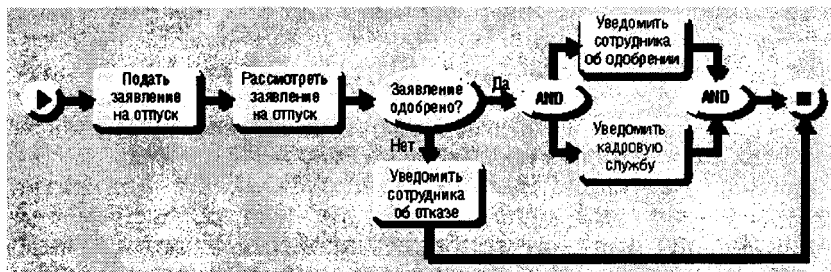
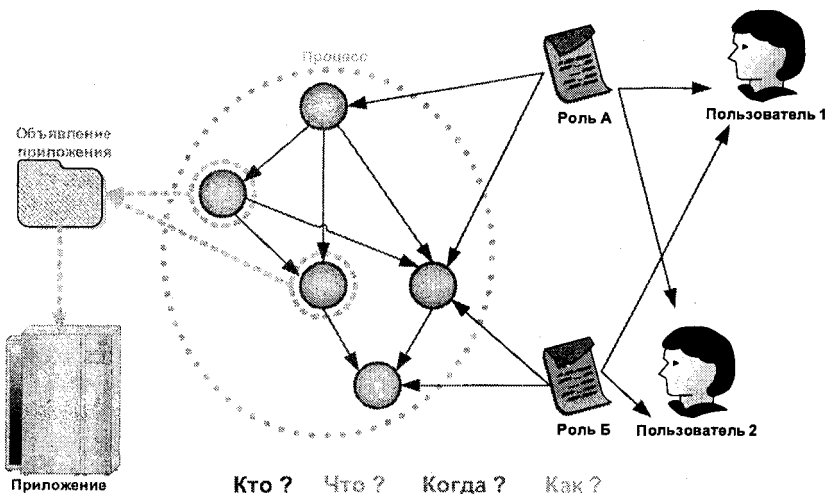


Рис. 1.11. Схема бизнес-процесса, соответствующего рассмотрению заявления об уходе в отпуск сотрудника предприятия

♦ обеспечивать средство формального единообразного описания различных бизнес-процессов на предприятии. На основе таких описаний формируется библиотека бизнес-процессов (бизнес-модель), управленцы проводят анализ и моделирование с целью повышения эффективности управления и гибкости своей организации;

♦ предоставлять возможность быстрой и динамической интеграции («склеивание») в рамках единого процесса действий различных сотрудников и компьютерных систем предприятия, а также позволять проводить быструю сборку из разнородных функциональных «кирпичиков» связного, качественного процесса.



**Рис. 1.12.** Основные элементы бизнес-процесса, учитываемые в языках для определения Workflow

Для переноса схемы бизнес-процесса в автоматизированную систему требуется описать бизнес-процесс формально: задать граф состояний, набор переменных, бизнес-правил и графических элементов форм, связать узлы графа состояний с соответствующими внешними приложениями или ролями пользователей и т.д. На этом этапе встает вопрос о формальных языках описания бизнес-процессов, которые позволяют перенести их в систему и дают точные ответы на вопросы «Кто? Что? Когда? Как?» (рис. 1.12).

В современной практике и теории автоматизации бизнес-процессов выделяют два типа языков для формального описания процессов и потоков задач (workflow):

◆ язык, разработанный для конкретной системы Workflow Management;

◆ язык, разработанный неким консорциумом как стандарт для реализации в целом классе систем.

В данной главе речь пойдет о втором типе языков. В основе большинства языков второго типа (как стартовая точка разработки концепции языка) лежит одна из двух хорошо известных математических теорий:

◆ теория сетей Петри [1, 2];

◆ аппарат математического исчисления Pi calculus [3].

Это позволяет разрабатывать строгие алгоритмы проверки непротиворечивости описаний процессов, а также проводить эффективное моделирование, анализ производительности и другие операции, которые требуются для повышения качества управления предприятием.

### *1.3.1. История стандартов WfMC на языки определения Workflow*

Консорциум WfMC (Workflow Management Coalition), образованный в 1993 г., был первым, предложившим спецификации для систем Workflow Management (и, в частности, спецификации на язык описания workflow), которые впоследствии несколько раз переписывались с учетом появления новых информационных технологий.

В 1999 г. был разработан язык определения бизнес-процессов WPDЛ (Workflow Process Definition Language). Его синтаксис был выражен в формулах Бэкуса-Наура. В 2002 г. язык WPDЛ был переписан. Его новая версия — XML Process Definition Language, XPDL — была основана на применении средств XML. Так, в частности, синтаксис языка XPDL полностью определяется соответствующей XML-схемой.

В марте 2004 г. WfMC призвала своих членов вносить предложения по расширению языка XPDL. В частности, предполагается внести в язык такие элементы, как таймеры, добавить в определение бизнес-процесса графические координаты узлов бизнес-процесса, а в определениях узлов графа бизнес-процесса — ссылки на связанные с ними пользовательские формы.

### 1.3.2. История стандартов и языки описания бизнес-процессов коалиции BPMI

В 2000 г. появилась коалиция BPMI, которая достаточно быстро разработала основанный на технологии Web-сервисов язык определения бизнес-процессов BPMML и начала создание других полезных стандартов (несовместимых со спецификациями WfMC).

Через некоторое время коалиция BPMI подготовила стандарт на язык BPMN, в котором был определен формат графических диаграмм для наглядного описания бизнес-процессов. Стандарт также содержал правила автоматического перевода графических диаграмм BPMN в язык BPMML.

Однако вскоре крупные компании IBM, Microsoft и BEA объединились в отдельный консорциум для создания другого языка, также основанного на технологии Web-сервисов (BPEL4WS<sup>1</sup>). В рядах коалиции BPMI началась паника — прогнозы абсолютного большинства экспертов сводились к тому, что предпочтение будет отдано именно этой спецификации и BPEL4WS станет стандартом де-факто в качестве языка определения бизнес-процессов.

Был период, когда BPMI позиционировала язык графических нотаций BPMN как графическую оболочку для BPEL4WS. Однако в настоящее время коалиция реанимировала BPMML и предлагает экспорт из графической нотации BPMN как в BPMML, так и в BPEL4WS. Но ситуация с языком BPMML остается очень неопределенной. Существует мнение, что BPMML проще и удобнее BPEL4WS, но вполне возможно, что корпорации-гиганты все-таки вытеснят его своей спецификацией BPEL4WS.

Ситуация с BPMN оказалась тоже далеко не безоблачной. Консорциум OMG разработал стандарт на диаграммы действий в языке UML 2.0 (Activity Diagrams). Они являются альтернативой языку BPMN (по выразительной графической силе эти нотации примерно одинаковы). Хотя многие считают, что для описания бизнес-процессов в настоящее время BPMN все же удобнее, чем Activity-диаграмма языка UML 2.0, однако вполне можно ожидать, что в следующей

---

<sup>1</sup> Об этом языке будет подробно рассказано в следующем разделе.



версии языка UML Activity-диаграмма вберет в себя все текущие преимущества BPMN и за счет маркетингового веса OMG именно она, а не BPMN, может стать фактическим стандартом графической нотации.

Коалиция BPMI создает также язык запросов для процессов (BPQL), который пока не привлек большого внимания.

### *1.3.3. История языка BPEL4WS*

К моменту образования коалиции BPMI корпорация IBM начала работу над своим стандартом языка для описания бизнес-процессов (WSFL), Microsoft также приступила к формированию собственной спецификации (XLANG; обе — несовместимы с XPDЛ и BPMЛ). В августе 2002 г. IBM, Microsoft и BEA объявили о подготовке совместного стандарта — языка BPEL4WS (или просто BPEL), позже к этим компаниям примкнули SAP и Siebel. При более близком знакомстве со стандартом возникает впечатление, что он неоправданно усложнен. Понятия, относящиеся к предметной области, находятся в нем на одном уровне с техническими понятиями, специфическими для технологии Web-сервисов. Это сильно ухудшает читаемость языка (например, по сравнению с XPDЛ). Системы Workflow Management легко стыкуются с Web-сервисами. Можно без труда определить интерфейс для взаимодействия с WF-системой через Web-сервисы, однако идея, предполагающая, что все взаимодействие с системами Workflow Management должно осуществляться только при помощи Web-сервисов, является сомнительной.

### *1.3.4. Другие стандарты*

К настоящему времени предложено большое количество других, косвенно относящихся к Workflow стандартов, пересекающихся и во многих случаях несовместимых со стандартами WfMC и BPMI. Вот некоторые из них:

- ◆ Business Process Specification Schema — BPSS (Electronic Business XML - ebXML) [www.ebxml.org/specs/ebBPSS.pdf](http://www.ebxml.org/specs/ebBPSS.pdf);

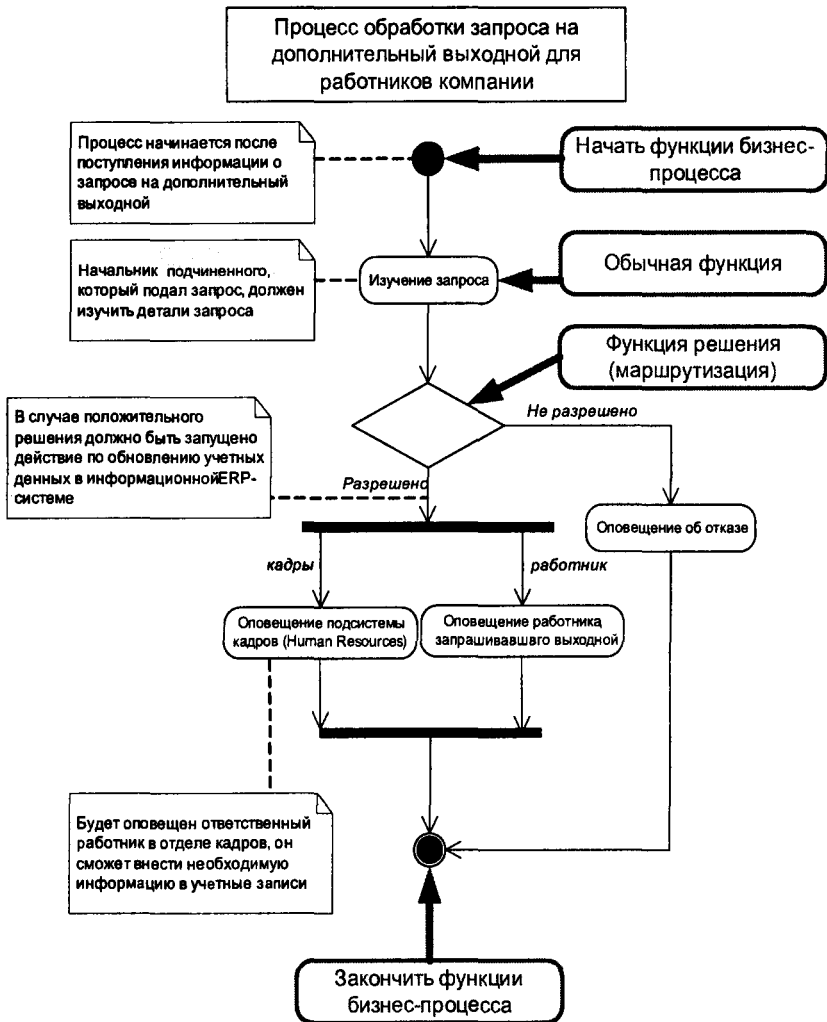
- ◆ Business Transaction Protocol — BTP (OASIS) [www.oasis-open.org/committees/download.php/1184](http://www.oasis-open.org/committees/download.php/1184);

- ◆ Web Services Conversation Language — WSCL (HP Labs/W3C) [www.w3.org/TR/2002/NOTE-wscl10-20020314](http://www.w3.org/TR/2002/NOTE-wscl10-20020314);
- ◆ Web Services Choreography Interface — WSCI (SUN/BEA/W3C). <http://ftpna2.bea.com/pub/downloads/wsci-spec-10.pdf>;
- ◆ Process Specification Language — PSL (National Institute of Standards and Technology, США) [www.mel.nist.gov/psl/](http://www.mel.nist.gov/psl/);
- ◆ Business Process Definition Metamodel (OMG) [www.bpmn.org/Documents/BPDM/OMG-BPD-2004-01-12-Revision.pdf](http://www.bpmn.org/Documents/BPDM/OMG-BPD-2004-01-12-Revision.pdf).

### *1.3.5. Тенденции развития стандартов на языки описания workflow*

Спецификации языков описания бизнес-процессов BPML и BPEL4WS были поданы в консорциум OASIS (Organization for the Advancement of Structured Information Standards) на утверждение в качестве промышленного стандарта. Также в OASIS была подана спецификация Wf-XML. Однако консорциум OASIS не утвердил в чистом виде ни BPEL4WS, ни BPML, а создал собственный комитет по разработке спецификации языка определения бизнес-процессов на основе BPEL4WS с учетом решений BPML. Эта спецификация будет называться WS-BPEL. Когда она будет разработана, неизвестно. Также вполне вероятно, что в будущем графическая спецификация BPMN сольется с диаграммами Activity языка UML, например, таким образом, как на рис. 1.13.

По сути, еще нет спецификации на язык описания бизнес-процессов, с которой не было бы связано серьезных проблем, даже лучшие из них пока выглядят неоправданно сложными. Возможно, реальным стандартом станет еще только разрабатываемая спецификация. Сложно указать современный прототип, на основе которого она будет создана. По-видимому, в условиях «войны стандартов» при выборе системы Workflow Management имеет смысл не привязываться к какому-либо одному стандарту, а предъявить менее жесткие требования, например обеспечение выполнения системой важных для данного предприятия Workflow patterns (набора образцов элементов бизнес-процессов, признанного международным сообществом).



**Рис. 1.13.** Пример описания Workflow с использованием средств языка UML

В нашей книге мы изучим подробно язык описания бизнес-процессов XPDL, предложенный консорциумом WfMC.

#### ***1.4. Неформальное введение в язык XPDL***

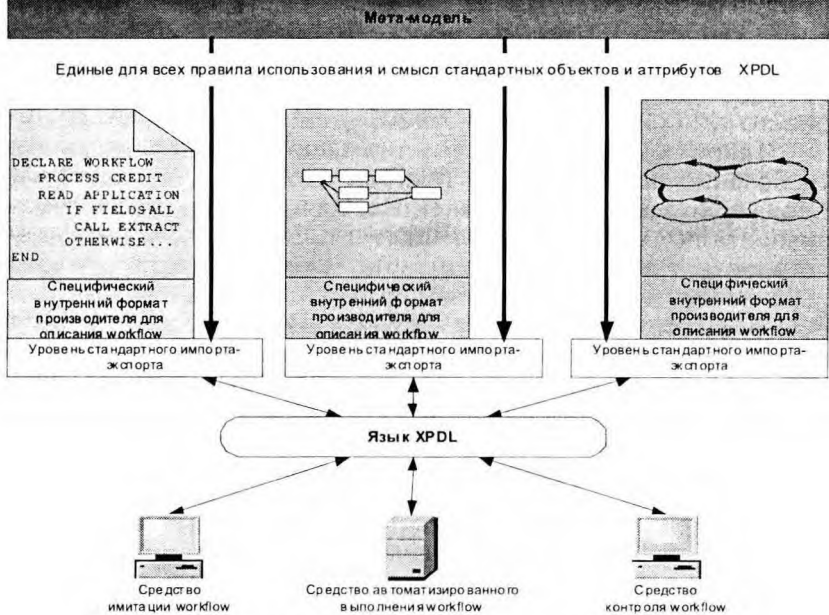
Как известно [7], консорциум WfMC в рамках своей деятельности по стандартизации технологии Workflow Management определил пять функциональных интерфейсов между различными компонентами «эталонной» системы. Спецификация языка для описания бизнес-процессов — XML Process Definition Language (XPDL) — является одной из составляющих интерфейса №1, определяя правила по экспорту-импорту определений бизнес-процессов между различными системами. В эту спецификацию входит формальное описание базовых конструкций языка, а также способ представления выражений на языке XPDL в формате XML (так называемая XML-схема).

Точное определение стандартного языка для описания бизнес-процессов оказывает большое влияние на широкое распространение технологии Workflow Management. Так как на практике для выявления, анализа и моделирования бизнес-процессов может использоваться большое число различных инструментов, то благодаря наличию единого стандарта на описание бизнес-процессов появляется возможность прозрачного переноса их определений из одного средства в другое. Можно указать на еще одно достоинство стандартного языка описания бизнес-процессов. В спецификации интерфейса № 1 четко выделяются две различные категории программных средств, относящихся к Workflow Management:

1) средства, предназначенные для разработки и моделирования описаний бизнес-процессов;

2) программные системы, которые служат для автоматизированного выполнения и контроля бизнес-процессов.

Использование одного и того же языка XPDL в программах, относящихся к этим двум категориям, позволяет описание бизнес-процесса, созданное в одном средстве для анализа и моделирования бизнес-процессов, использовать для реального выполнения процессов в различных систе-



**Рис. 1.14.** Концепция единого языка описания workflow на основе мета-модели

мах, которые обеспечивают автоматизированное выполнение и контроль бизнес-процессов. Базовые конструкции языка XPDL строго определяются с использованием так называемой мета-модели. Мета-модель является формальным языком, который однозначно определяет все сущности, используемые в процессе описания workflow на языке XPDL, взаимосвязи между сущностями, а также стандартные атрибуты, которые определяют характеристики этих сущностей в явном виде.

Именно мета-модель является основой единого метода доступа и описания определений workflow. Благодаря наличию общей мета-модели программные средства различных производителей могут производить обмен описаниями через общий формат обмена (рис. 1.14). При этом в языке XPDL разрешается использовать собственные расширения производителей, известные ограниченному числу инстру-

ментальных средств. Такие расширения реализованы в виде произвольного множества дополнительных атрибутов стандартных элементов языка XPDЛ. Те программы, которые не обладают информацией о расширенном наборе атрибутов, просто игнорируют их в процессе чтения XPDЛ-файла.

Такой подход налагает минимальные дополнительные требования на реализацию программных средств различных производителей, предназначенных для описания и автоматизированного выполнения бизнес-процессов. Чтобы удовлетворить требованиям стандарта, производители должны реализовать в своих средствах всего лишь две операции:

- ◆ импорт определения workflow во внутренний формат из стандартного формата XPDЛ;
- ◆ экспорт определения workflow из внутреннего формата в стандартный формат XPDЛ.

### *1.4.1. Структура языка XPDЛ*

Как уже упоминалось ранее, синтаксис языка XPDЛ основан на XML. Это означает, что описание бизнес-процесса на языке XPDЛ представляется в виде текстового XML-документа, а разрешенный набор элементов (тегов) и их взаимосвязи определяются особой XML-схемой. Кроме того, существует графическая форма представления основных элементов языка XPDЛ.

Основными элементами языка XPDЛ являются:

- ◆ пакет (Package);
- ◆ приложение (Application);
- ◆ workflow-процесс (Workflow-Process), или бизнес-процесс;
- ◆ функция (Activity);
- ◆ переход (Transition);
- ◆ участник (Participant);
- ◆ поле данных (DataField);
- ◆ тип данных (DataType).

Элемент «Пакет» является контейнером, содержащим все остальные элементы. Элемент «Приложение» используется для спецификации приложений (программ, инструментов, устройств и т.п.), вызываемых в ходе выполнения бизнес-процессов, определенных в пакете. Элемент

«workflow-процесс» применяется для точного определения порядка выполнения процессов или их отдельных частей. В языке XPDЛ определение workflow-процесса состоит из комбинации элементов «Функция» и «Переход». Элемент «Функция» является основным строительным блоком определения процесса. В языке XPDЛ выделяются три типа функций:

- ◆ маршрутизация (Route);
- ◆ исполнение (Implementation);
- ◆ группа (BlockActivity).

Функции типа «Маршрутизация» реально никаких действий не выполняют, а служат для определения порядка выполнения отдельных функций в workflow-процессе. Функция типа «Исполнение» является отдельным шагом процесса. Эта функция выполняется либо вручную, либо одним или несколькими приложениями, либо другим workflow-процессом (случай так называемого подчиненного процесса, в оригинале — subflow). Функции типа «Группа» служат для совместного выполнения группы подчиненных функций. Особый вспомогательный элемент «Набор функций» (ActivitySet) определяет самодостаточный набор функций и переходов. Обычно набор функций используется для определения конкретного содержимого элемента «группа» (рис. 01.15).

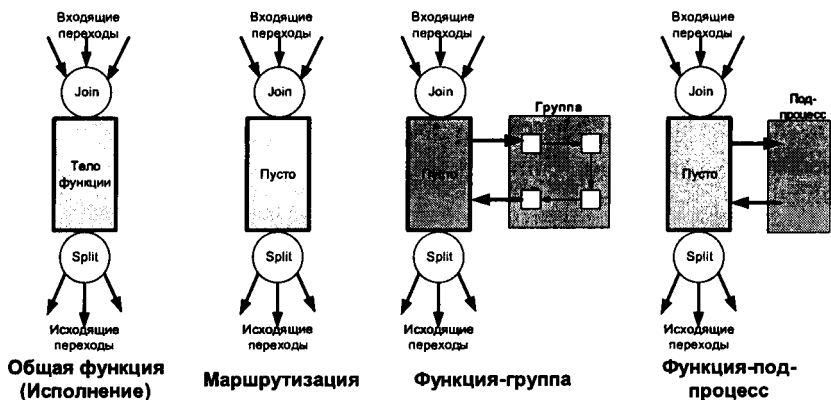


Рис. 1.15. Определение различных типов функций в языке XPDЛ

Нужно отметить, что согласно спецификации XPDЛ реально существует только один XML-элемент, обозначающий абстрактную функцию. Отдельные элементы для конкретных типов функций (implementation, route, block) отсутствуют. Дифференциация конкретных различных типов функций производится при помощи особых атрибутов элемента. Эти атрибуты носят соответственно названия Route, Implementation и BlockActivity.

В определении функции дополнительно может указываться уровень автоматизации этой функции (автоматизированное или ручное выполнение). Выполнение автоматизированной функции может полностью контролироваться системой Workflow Management с использованием встроенных или внешних программных компонентов. Ручные функции требуют вмешательства людей. Между собой функции соединяются с помощью элементов «Переход», образуя единую структуру в виде графа — описание workflow-процесса. Вершинами этого графа являются функции, а ребра определяются переходами. Каждый переход (ребро) соединяет ровно две вершины (функции). У элемента «Переход» есть два обязательных атрибута, From и To, в которых содержатся ссылки соответственно на функцию-источник и функцию-адресат. После завершения функции-источника происходит «срабатывание» перехода и запускается на выполнение функция-адресат. При определении перехода можно указать особые условия его срабатывания. Для этого используется атрибут перехода «условие» (Condition). Существует четыре вида условий:

- ◆ **CONDITION**: переход может сработать, если логическое выражение, определенное в атрибуте «условие», является истинным;

- ◆ **OTHERWISE**: переход может сработать, если ни в одном из других переходов логическое выражение не является истинным (см. далее);

- ◆ **EXCEPTION**: исключение (exception) является особым типом перехода, который срабатывает в случае необычного поведения (например, деление на ноль);

- ◆ **DEFAULTEXCEPTION**: исключение по умолчанию. Переход срабатывает при возникновении необычной ситуа-



ции, которая не была обработана другими переходами с типом условия EXCEPTION.

Как видно из рис. 1.5, каждая функция является точкой слияния нескольких входящих переходов — Join (можно также сказать — точкой слияния нескольких параллельно выполняющихся функций) и одновременно точкой, где маршрут workflow-процесса расщепляется на несколько исходящих переходов — Split (несколько параллельно запускаемых функций). Механизм слияния и расщепления позволяет описывать различные правила параллельного выполнения нескольких функций и процессов. Для этого в ходе определения функции можно задавать одно или несколько «ограничений на переход» (в оригинале — «transition restrictions»). Эти ограничения однозначно определяют принципы запуска функции, когда она является точкой слияния нескольких переходов или когда после завершения функции происходит расщепление на несколько переходов.

Если функция имеет ограничение на «слияние» (Join), то оно может принимать одно из двух значений:

- ◆ И (Join-AND);
- ◆ ИСКЛЮЧАЮЩЕЕ-ИЛИ (Join-XOR).

Вот как определяется смысл ограничения Join-AND в стандарте WfMC: «Происходит слияние всех параллельно выполняющихся функций в точке Join данной функции. При этом обязательна синхронизация — переход не срабатывает, а данная функция не запустится на исполнение, пока не закончится выполнение всех сливаемых функций». А вот так определяется смысл ограничения Join-XOR: «Происходит слияние какой-либо одной из нескольких параллельно выполняющихся функций в точке Join данной функции. При этом синхронизации не требуется<sup>4</sup>».

Соответственно, существует ограничение на «расщепление» (Split), которое может принимать значения:

- ◆ И (Split-AND);
- ◆ ИСКЛЮЧАЮЩЕЕ-ИЛИ (Split-XOR).

Split-AND: «Одновременно запускается несколько функций, соединенных выходными переходами с текущей фун-

---

<sup>4</sup> Иными словами, для срабатывания перехода и запуска функции достаточно завершения всего лишь одной сливаемой функции.

кцией в точке Split». Split-XOR: «Из нескольких функций, соединенных выходными переходами с текущей функцией в точке Split, запускается только одна. То, какую функцию запускать, определяется в результате последовательной проверки условий в соответствующих переходах. Как только находится первый переход, у которого условие истинно, дальнейшая проверка прекращается. В перечне могут находиться переходы без проверок, или функции с условием OTHERWISE. Они запускаются в том случае, если ни один из переходов не сработал».

Параллельное выполнение нескольких функций начинается Split-AND и заканчивается Join-AND (рис. 1.16).

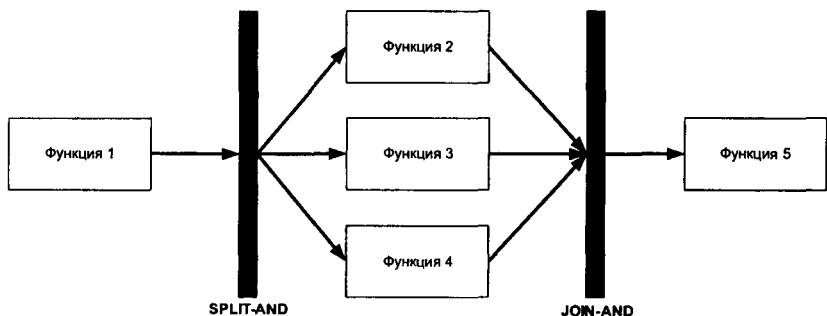


Рис. 1.16. Преставление расщепления-слияния и параллельного выполнения функций

При определении функций и переходов можно указывать также ограничения, распространяющиеся на структуру всего графа функций и переходов. Эти ограничения носят название «классы соответствия» (в оригинале — conformance classes). Каждый класс соответствия определяет свой набор ограничений, которым должен удовлетворять граф функций и переходов:

◆ Класс соответствия NON-BLOCKED разрешает строить граф с произвольной структурой (с точки зрения соответствия количества расщеплений и слияний, а также наличия циклов).

◆ Класс соответствия LOOP-BLOCKED требует, чтобы в графе не было циклов (т.е. граф должен принадлежать к классу ориентированных ациклических графов).

◆ Класс соответствия FULL-BLOCKED требует, чтобы каждому элементу split-AND соответствовал ровно один элемент join-AND, каждому элементу split-XOR соответствовал ровно один элемент join-XOR. Дополнительно каждый маршрут, начинающийся с расщепления, должен достигать соответствующего слияния.

Элемент «Участник» используется для определения участников бизнес-процесса, т.е. отдельных сущностей, способных выполнять определенные задания. В языке XPDЛ определяется шесть типов участников:

- 1) набор ресурсов (ResourceSet);
- 2) ресурс (Resource);
- 3) роль (Role) — соответствует определенной должности в организационной структуре предприятия;
- 4) подразделение организации (OrganizationalUnit) — не обязательно отдельная должность, может ссылаться на целый отдел, рабочую группу, цех и т.п.;
- 5) человек (Human) — конкретный исполнитель (например, «Иван Иванов, табельный номер 12334»), взаимодействующий с системой Workflow management;
- 6) система (System) — программный компонент, использующийся для выполнения полностью автоматизированной функции.

Участник представляет из себя «уровень абстракции, отделяющий реального исполнителя от функции, которую он исполняет». Поэтому во время определения workflow-процесса на языке XPDЛ задаются лишь абстрактные участники, принадлежащие к одному из шести типов. Абстрактные участники идентифицируются уникальным именем, которое не связано с конкретными фамилиями работников предприятия, названием программы и т.п. Назначение абстрактных участников на выполнение функций производится указанием их уникального имени в атрибуте Performer соответствующего элемента «Функция». Перед началом выполнения workflow-процесса в системе Workflow Management обязательно нужно привести в соответствие каждому абстрактному участнику, указанному в описании, реального

работника предприятия, конкретную программную систему с заданным набором стартовых параметров и т.п. Соответствие устанавливается с помощью служебных конфигурационных программ, входящих в состав конкретной системы Workflow Management. Нужно сказать, что подобно участникам, элемент «Приложение» на языке XPDЛ также определяется с использованием абстрактного имени приложения, его типа и списка формальных параметров. Имя приложения является всего лишь уникальным идентификатором, который необязательно соответствует реальному физическому расположению приложения и деталям реализации. Поэтому перед запуском процесса нужно провести операцию сопоставления абстрактных имен приложений реальным программным реализациям. Элементы типа «Поле данных» и «Тип данных» используются для определения специфичных для workflow данных (в оригинале — workflow relevant data). В ходе выполнения workflow-процесса они передаются между функциями и подчиненными процессами и используются для выработки решений по маршрутизации или ссылаются на внешние источники данных.

#### *1.4.2. Состав мета-модели языка XPDЛ*

Все перечисленные в предыдущем разделе элементы формируют мета-модель языка XPDЛ. Она может быть представлена в виде следующей диаграммы классов UML (рис. 1.17).

Русское название основных элементов языка XPDЛ, присутствующих на рис. 1.17, и их назначение представлены в табл. 1.1.

Мета-модель (и, соответственно, язык XPDЛ) предполагает использование стандартных типов данных (строковый тип, ссылка, целое, вещественное, дата/время и т.п.) для описания данных, относящихся к workflow, системных данных и информации об окружении, а также данных об участнике workflow. С использованием стандартных типов данных могут быть построены условные выражения, определяющие последовательность выполнения функций. Кроме того, набор стандартных типов может быть расширен пользователем (для этого используется XML-схема или ссылка на данные, определенные в другом месте).

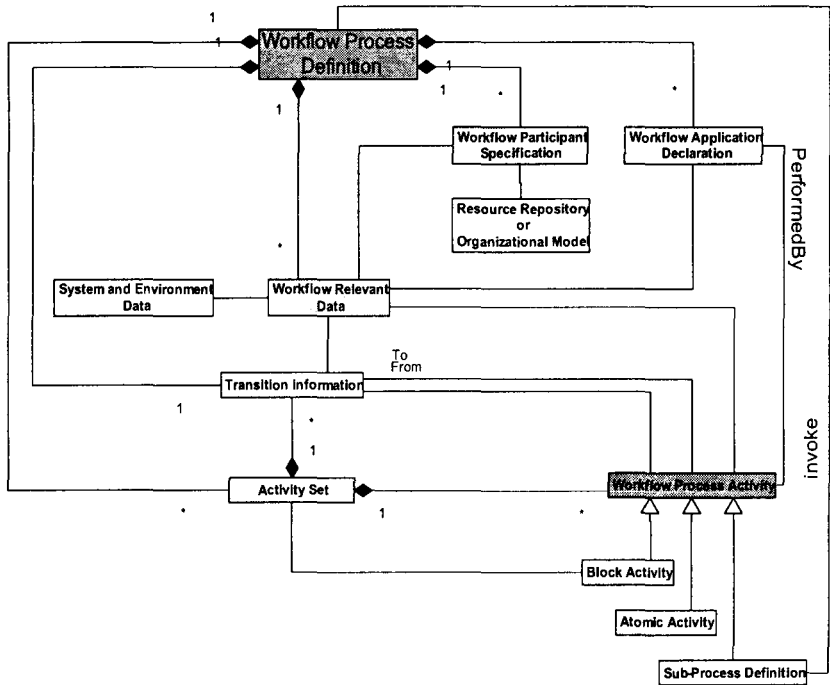


Рис. 1.17. Мета-модель определения процесса Workflow

Из определения мета-модели видно, что модель процесса включает в себя различные сущности, один и тот же экземпляр которых может встречаться в определении нескольких различных бизнес-процессов (рис. 1.18). В частности, ссылки на определение одного и того же участника, приложение или данные, относящиеся к workflow, могут присутствовать в описании независимых различных процессов. Чтобы сократить избыточность данных, при использовании мета-модели языка XPDЛ предполагается применение единого репозитория определений workflow, связанного с определенной системой Workflow Management. В этом репозитории хранится описание различных типов сущностей, составляющих описания workflow. Для структу-

## Основные сущности мета-модели языка XPDL

№	Английское название	Русское название	Значение
1	Workflow Process Definition	Определение процесса	Контекстная информация, являющаяся входом для всех процессов. Является контейнером для процесса
2	Workflow Process Activity	Функция процесса	Описание процесса включает, в том числе, и действия внутри процесса, выполняемые за счет комбинации ресурсов
3	Transition Information	Информация о переходе	Действия взаимодействуют друг с другом с учетом условий (информации) о переходах
4	Workflow Participant Declaration	Объявление участника Workflow	Обеспечение описания участника процесса
5	Resource Repository	Хранилище ресурсов	Обеспечение хранения разнородных ресурсов
6	Workflow Application Declaration	Объявление приложения Workflow	Описание приложения или интерфейса
7	Workflow Relevant Data	Данные, относящиеся к Workflow	Описание данных
8	System and Environmental Data	Системные данные и информация об окружении	Описание данных

рирования данных внутри репозитория и обеспечения эффективных процедур импорта/экспорта описаний в/из репозитория используется понятие пакета (package).

Пакет играет роль контейнера, в котором сгруппированы определения сущностей, общих для нескольких различных определений workflow. Это позволяет не повторять переопределение в каждом отдельном процессе. Пакет обеспечивает контейнер, в котором хранится ряд общих атрибутов, определяющих «родовой» процесс (автор, версия, статус и т.д.). Описание каждого конкретного процесса, содержащееся в пакете, будет автоматически наследовать значение любого общего атрибута. При этом значение атрибута может быть переопределено локально, в рамках определения отдельного пакета.

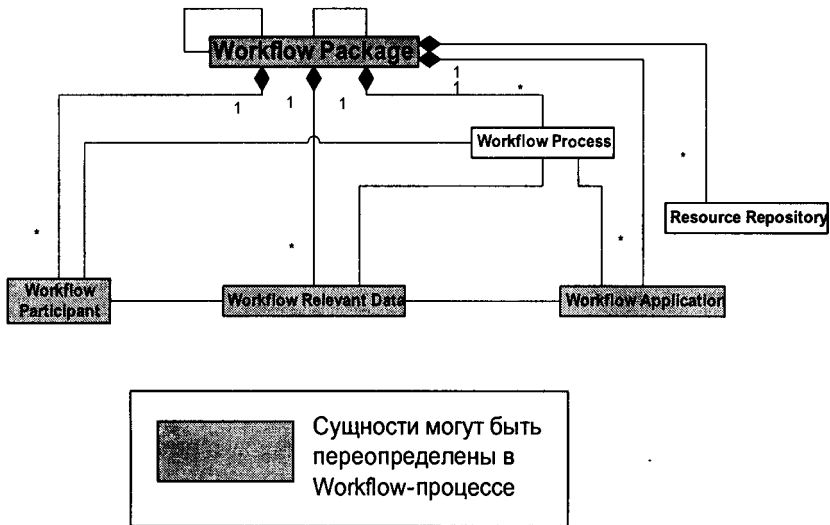


Рис. 1.18. Мета-модель для определения пакета

Внутри пакета определение некоторых сущностей имеет глобальную область видимости. Это означает, что на эти определения можно ссылаться из любого описания бизнес-процесса (а также связанных функций и переходов), находящегося в данном пакете. К таким сущностям с глобальной областью видимости относятся:

- ◆ определение участников workflow;
- ◆ объявление приложения workflow;
- ◆ данные, относящиеся к workflow.

Можно использовать также ссылки на внешние пакеты. Эти ссылки позволяют использовать внутри текущего пакета и содержащихся в нем объектов глобальные сущности, определенные в пакете, на который указывает ссылка. К таким глобальным сущностям относятся:

- ◆ идентификаторы процессов (используются для организации subflow);
- ◆ определение участников workflow;
- ◆ объявление приложения workflow.

### 1.4.3. Описание атрибутов стандартных элементов языка XPDЛ

В табл. 1.2 представлены стандартные атрибуты элементов языка XPDЛ, определенных в мета-модели. Атрибуты (строки таблицы) разбиты на пять групп.

◆ Первая группа содержит атрибуты, общие для всех важнейших элементов языка XPDЛ. Так, все важнейшие элементы XPDЛ содержат атрибуты *id* и *name*, а также могут иметь атрибут *Description* и набор расширенных атрибутов (*Extended Attributes*).

◆ Вторая группа включает атрибуты, которые относятся лишь к определенному типу элемента.

◆ Третья группа состоит из атрибутов, которые могут содержать ссылки на другие элементы.

◆ Четвертая группа содержит особые атрибуты *Documentation* и *Icon*, которые служат для хранения информации о внешнем виде элемента для системы автоматизированного выполнения бизнес-процесса.

◆ Пятая группа — это атрибуты, имеющие отношение к проведению имитационных экспериментов и оптимизации бизнес-процессов (так называемая *BPR-relevant information*).

### 1.4.4. Пример описания workflow на языке XPDЛ

В качестве примера приведем описание бизнес-процесса рассмотрения заказа (*Order*) на покупку определенного товара. При описании бизнес-процессов будет использоваться графическая нотация для элементов языка XPDЛ, представленная на рис. 1.19.

#### Процессы

##### 1. Главный процесс ввода заказа (*EOrder Main Process*)

Главный процесс на входе получает отформатированную по особым правилам текстовую строку и возвращает строку, которая информирует о том, был ли заказ принят (*Confirmed*) или отклонен (*Rejected*). В этом процессе выделяются следующие шаги.



## Основные атрибуты сущностей в мета-модели языка XPDL

Package (Пакет)	Workflow Process (Процесс Workflow)	Activity (Функция)	Transition (Переход)	Application (Приложение)	Data Field – Workflow Relevant Data (Поле данных)	Participant (Участник)
Id	Id	Id	Id	Id	Id	Id
Name	Name	Name	Name	Name	Name	Name
Description	Description	Description	Description	Description	Description	Description
Extended Attributes	Extended Attributes	Extended Attributes	Extended Attributes	Extended Attributes	Extended Attributes	Extended Attributes
XPDL Version	Creation Date	Automation Mode			Data Type	Participant Type
Source Vendor ID	Version	Split				
Creation Date	Author	Join				
Version	Codepage	Priority				
Author	Country Key	Limit				
Codepage	Publication Status	Start Mode				
Country Key	Priority	Finish Mode				
Publication Status	Limit	Deadline				
Conformance Class	Valid From Date					
Priority Unit	Valid To Date					
Responsible	Parameters	Performer	Condition	Parameters	Initial Value	
	Responsible	Tool	From			
		Subflow	To			
		ActivitySet				
		Actual Parameter				
External Package						
Documentation	Documentation	Documentation				
Icon	Icon	Icon				
Cost Unit	Duration Unit	Cost				
	Duration	Duration				
	Waiting Time	Waiting Time				
	Working Time	Working Time				






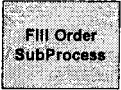
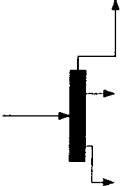
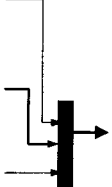
№	Графический вид	Смысл конструкции
1		Начало бизнес-процесса
2		Конец бизнес-процесса
3		Функция типа «исполнение». Функцию исполняет внутреннее приложение или какой-либо участник.
4		Функция типа «маршрутизация».
5		Функция типа «исполнение». Функцию исполняет внешнее приложение.
6		Функция типа «группа». Начинается либо набор функций, либо отдельный подчиненный процесс.
7		Ограничение на переход типа «Split-AND».
8		Ограничение на переход типа «Join-AND».

Рис. 1.19. Используемая графическая нотация

1. Преобразование текстовой строки в сложный объект данных. Если в процессе преобразования возникает исключительная ситуация (exception), оповещающая о том, что строка сформирована неправильно, то выдается сигнал ошибки (alarm) и заказ отклоняется.

2. Данные заказа проверяются на логическую непротиворечивость.

3. Определяется, каким образом будет проходить оплата: по счету (purchase order, PO) или банковской картой (credit card).

4. Заказ, оплачиваемый банковской картой, посылается в подпроцесс авторизации оплаты по карте.

5. Заказы, оплачиваемые по счету, контролируются приложением, которое проверяет данные продавца о поступлении нужной суммы денег по счету.

6. Заказ помещается в базу данных и ему присваивается номер.

7. Следующие три функции выполняются параллельно:  
а) формируется сообщение о принятии заказа для пользователя (заказчика);

б) вызывается подпроцесс для заполнения всех деталей по данному заказу;

в) уведомление о принятии заказа пересылается по электронной почте пользователю (заказчику). Эта функция имеет особенность – используются расширенные атрибуты, которые определяют дополнительные параметры, требующиеся для отправки электронной почты (например, адрес сервера, пароль и т.п.).

8. Если заказ был отклонен либо по причине логической противоречивости, либо в силу невозможности авторизации, то формируется и отправляется сообщение для пользователя.

Описанный процесс в графическом виде представлен на рис. 1.20.

## *2. Подпроцесс проверки банковской карты (The CreditCheck Subprocess)*

Подпроцесс проверки банковской карты (CreditCheck subprocess) инициализирует поля служебного объекта CreditInfo входными параметрами и пересылает информа-

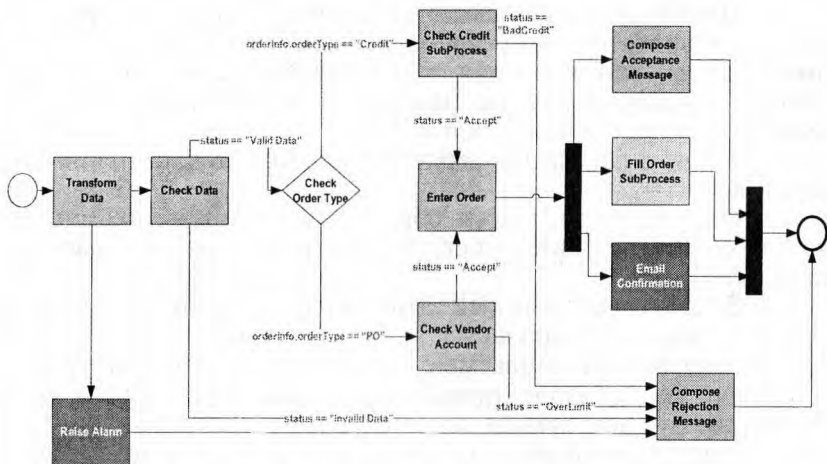


Рис. 1.20. Процесс обработки заказа

цию внешнему Web-приложению для авторизации. Внешнее Web-приложение возвращает строку со статусом проверки. Эта строка конвертируется в объект-строку OrderStatus и возвращается в качестве окончательного результата вызывающему процессу.

Описанный процесс в графическом виде представлен на рис.1.21.



Рис. 1.21. Подпроцесс проверки банковской карты

### 3. Подпроцесс заполнения деталей заказа (The FillOrder Subprocess)

В этом подпроцессе производится заполнение служебной формы на доставку товара (Shipping Info) и полной отчетности по оплате заказа для пользователя (Billing Info).

Этот подпроцесс включает в себя участника, названного «Курьер» («Shipper»).

1. Первая функция подготавливает информацию по заказу для Курьера, который будет доставлять товар заказчику. Приложение возвращает статус заказа — был ли он завершен или задержан с исполнением. В выполнении этой функции установлена критическая дата. Если функция не будет успешно выполнена в течение трех дней, то возникает исключительная ситуация (Exception) notifyException и выдается сигнал ошибки (alarm). Если функция не может быть завершена успешно в течение пяти дней, то возникает исключительная ситуация timeoutException и заказ аннулируется.

2. Затем подпроцесс определяет, каким образом проводится оплата заказа (PO or credit card order).

3. Заказы, оплачиваемые по счету (PO), посылаются в биллинговую систему, а затем создается электронный счет (invoice), сохраняемый на сервере.

4. Заказы, оплачиваемые по банковской карте, пересылаются во внешнее приложение по обслуживанию банковских карт для снятия денег, а затем создается электронный чек (electronic receipt), сохраняемый на сервере.

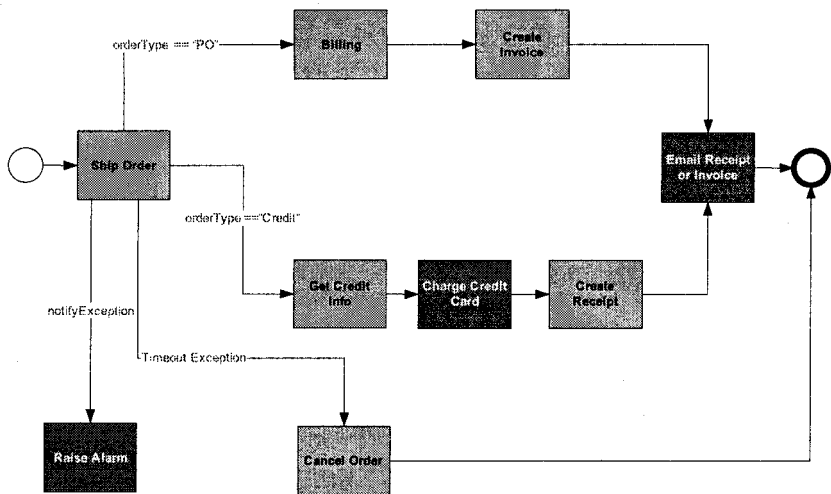


Рис. 1.22. Подпроцесс заполнения деталей заказа

5. Завершающая функция подпроцесса пересылает счет или чек пользователю в виде вложения в электронное письмо. При этом используются расширенные атрибуты, определяющие необходимые для отправки почты параметры.

Описанный процесс в графическом виде представлен на рис. 1.22.

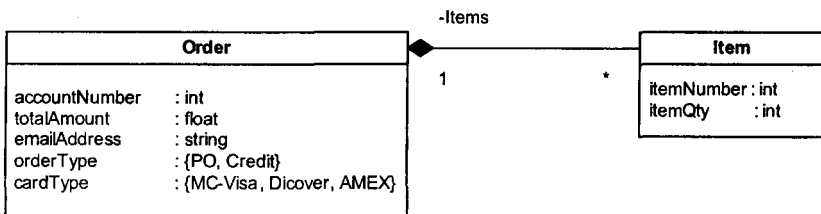
## 1.5. Определения типов данных, используемых в бизнес-процессе

Полное описание всех рассмотренных бизнес-процессов и типов данных, а также описание расширенных атрибутов и объявление приложений в форме XML-документа приведены в документе TC-1025\_10\_xpdl\_102502.pdf (Стандарт на язык описания workflow XPD<sup>L</sup>), доступном на сайте [www.wfmc.org](http://www.wfmc.org).

В этой главе мы приводим лишь краткую справку по используемым структурам данных в форме UML-диаграмм.

### Order

```
<TypeDeclaration Id="Order" Name="Order">
<ExternalReference
location= "http://wfmc.org/standards/docs/xpdl_sample/
orderschema.xsd"/>
</TypeDeclaration >
```



### OrderStatus

Type Status: {ValidData, InvalidData, Accept, Bad Credit, OverLimit, BadDataFroma}

## *CardType*

При определении этого типа данных используется ссылка на определение типа *Order*.

## *CreditInfo*

<b>CreditInfo</b>
MerchantNumber
AccountNumber
Amount
CardType

### ***1.6. Приложение к гл. 1. Математические основы автоматизированного управления бизнес-процессами***

#### ***1.6.1. Теория сетей Петри***

Эта математическая дисциплина, основанная на классической теории графов, является расширением теории конечных автоматов. Она возникла в 60-х гг. XX в. и с тех пор постоянно развивается. На основе одного из вариантов сетей Петри (*High-level Petri Nets*) в настоящее время создается международный стандарт ISO/IEC 15909 [4]. Сети Петри — сложная, очень хорошо развитая теория, в ней строго определены такие понятия, как состояния, условия, переходы и т. п. Она включает также графическую нотацию (систему графических обозначений, с помощью которых можно рисовать соответствующие графы). Эта область хорошо исследована математиками: установлены многие свойства сетей Петри, доказаны важные теоремы.

Практическое использование теории сетей Петри в основном было связано с описанием поведения очень сложных систем, например, элементов интегральных схем. Построив для системы сеть Петри, далее можно было использовать развитый математический аппарат и таким образом исследовать ее свойства. Для описания WF-систем использовать концепцию сетей Петри в явном виде неудобно, так как принятая там графическая нотация не является интуитивно понятной (бизнес-аналитикам, а тем более менеджерам с ней сложно работать) и, кроме того, не все

виды бизнес-процессов можно описать с ее помощью. Наследниками теории сетей Петри стали первые языки описания процессов (например, WPDЛ и XPDЛ коалиции WfMC). Они основаны на теории графов и включают в себя многие понятия и концепции сетей Петри — узлы, переходы, условия и т. д., однако в отличие от сетей Петри эти языки не являются строгими: в ряде случаев можно составить такие языковые конструкции, которые будут синтаксически допустимыми, хотя поведение порожденного ими бизнес-процесса не будет определено однозначно.

Аалст и Хофстед предложили расширение концепции сетей Петри для систем Workflow Management, введя дополнительные базовые элементы. Этот подход привел к появлению нового строгого языка — YAWL (Yet Another WorkFlow Language, «еще один WorkFlow-язык») [5]. К сожалению, он оказался очень сложным, его графические диаграммы также непонятны интуитивно, и, скорее всего, использоваться он будет исключительно в теоретических целях.

### *1.6.2. Ограниченность WF-языков, основанных на теории сетей Петри*

Эта ограниченность — следствие того, что концепция сетей Петри основана на теории графов. В последнее время в программировании предложены понятия, не укладывающиеся в рамки теории графов, например, исключения (exceptions). Эти новые «программистские» понятия были применены при разработке некоторых WF-языков (в частности, BPMЛ/BPMN) и оказались очень полезными. Таким образом, роль теории сетей Петри в мире WF-языков неоднозначна: с одной стороны, эту теорию можно использовать для исследования некоторых видов WF-процессов, с другой — с ее помощью нельзя описать все процессы. Кроме того, диаграммы сетей Петри чрезвычайно громоздки: в случае сложных процессов соответствующие им сети Петри содержат огромное количество элементов, и разобраться в них очень трудно.

### *1.6.3. Концепция Pi calculus*

Концепция Pi calculus ( $p$ -исчисление) была разработана в конце 80-х гг. XX в. Робинот Милнером и основана на



алгебре параллельных процессов. В отличие от сетей Петри математическими объектами  $\rho$ -исчисления являются не графы, а выражения над элементами специальных множеств и преобразования этих выражений. В настоящее время  $\rho$ -исчисление является перспективной, хотя и очень молодой и развивающейся теорией, в ней много открытых вопросов и нерешенных проблем.

Разработчики двух из наиболее интересных в настоящее время языков — BPML и BPEL4WS — утверждают, что, так как в их основе лежит солидная математическая теория —  $\rho$ -исчисление, эти языки обладают очень высокой выразительной мощностью. Однако немало и скептиков, считающих, что связь этих языков с концепцией Pi calculus не очевидна, и предполагающих, что мы имеем дело скорее с маркетинговым ходом, чем с реальным использованием сложной математической теории для построения WF-языка [6].

#### *1.6.4. Понятия модели и мета-модели, их совместное использование в современных информационных технологиях*

Моделирование является одним из наиболее распространенных способов изучения различных сложных систем. Человек постоянно строит модели, чтобы сократить число одновременно рассматриваемых явлений, предсказать поведение системы и т.п. Каждая модель использует особый язык для определения структуры и поведения, наиболее приспособленный для решения ограниченного круга задач. Современные программные системы достигли такого уровня сложности, когда глубокое изучение программной архитектуры, алгоритмов и принципов реализации уже не может проводиться без предварительной подготовки. В мире программ стало необходимым использование механизма моделей, которые дают возможность разработчикам абстрагироваться от несущественных на данный момент деталей реализации и сконцентрироваться на понимании основополагающих понятий, без которых невозможна сама работа программной системы.

На модель можно смотреть как на удобный язык описания, который определяет структуру и поведение опре-

деленной системы, гораздо понятней и точней, чем естественный язык. Но, как и в обычном языке, здесь возникает проблема — как гарантировать, что описание системы на языке модели было одинаково понято различными пользователями (причем не только людьми, но и компьютерными программами)? Если не решать этот вопрос, то можно оказаться в ситуации, когда слова и выражения языка модели интерпретируются по-разному различными пользователями. Такую ситуацию демонстрирует фрагмент из произведения Л.Кэрролла:

«...»

— Но «огород» — вовсе не значит «славенький сногшибательный аргументик», — возразила Алиса.

— Когда лично я употребляю слово, — все так же презрительно проговорил Шалтай-Болтай, — оно меня слушается и означает как раз то, что я хочу: ни больше, ни меньше.

— Это еще вопрос, — сказала Алиса, — захотят ли слова вас слушаться.

— Это еще вопрос, — сказал Шалтай, — кто здесь хозяин: слова или я.

...»

Льюис Кэрролл,  
«Алиса в Зазеркалье»

Чтобы смысл отдельных структурных элементов и всей модели целиком был одним и тем же для различных пользователей, необходимо обладать общедоступным формальным описанием структурных элементов, из которых может состоять любая правильная модель. Прибегая к аналогии с языком, для этого нужно создать еще один особый язык, на котором можно строго описать смысл слов и выражений на языке модели. Чтобы подчеркнуть особое положение этого второго языка по отношению к исходному языку модели используется приставка «мета-», которая указывает на то, что «мета-язык» находится выше в иерархии по отношению к исходному понятию «язык».

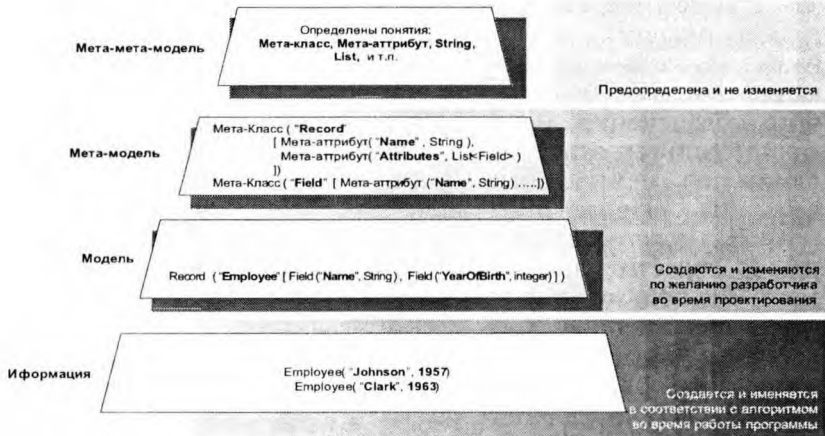
Таким образом, мета-язык служит для создания мета-модели, которая позволяет различным пользователям (и программам) одинаково и точно понимать смысл моделей.

Процесс создания взаимосвязанной иерархии моделей и мета-моделей для точного понимания смысла носит название мета-моделирования (рис. 1.23). Мета-моделирование известно не только в области разработки программного обеспечения, но и в лингвистике, психологии и т.п. Достаточно сказать, что понятие «мета-модель» является одним из центральных в популярном психологическом течении «Нейро-лингвистическое программирование» (<http://ptu-z.narod.ru/lib/nlp/gl5.html>).

В области разработки программного обеспечения активное использование концепции иерархии моделей и мета-моделирования началось с конца 80-х гг. С ростом популярности Интернета и повышением важности задач по интеграции разнородных данных мета-моделирование, безусловно, стало наиболее важным направлением современной информатики и программной инженерии. Это обусловлено тем, что мета-модели являются основой для автоматизированной интеграции разнородных данных.

Как известно, мета-модель является формальным языком, который используется для спецификации моделей. Мета-модель содержит описание всех конструктивных элементов, которые разрешено использовать при создании конкретной модели. Таким образом, модели являются экземпляром мета-модели. Мета-модель обычно содержит перечень типов разрешенных конструктивных элементов вместе с атрибутами, описание разрешенных типов связей между различными строительными элементами и другую полезную информацию. Все это вместе позволяет однозначно и в доступной для человека и компьютера форме определить смысловое значение (семантику) каждого конструктивного элемента в отдельности и модели в целом. Сами модели, в свою очередь, определяют структуры данных, реально использующиеся в приложении. Так, в объектно-ориентированном программировании реальные объекты в программе являются экземплярами классов модели. Поэтому модели можно назвать мета-данными (т.е. информацией о свойствах и структуре реально используемых данных).

Проиллюстрировать взаимоотношение между разными уровнями мета-моделирования можно на основе стандартов международной организации OMG<sup>5</sup>. Для моделирования и



**Рис. 1.24.** Пример конкретной реализации иерархии MOF (жирным шрифтом на каждом уровне иерархии MOF обозначены вводимые на этом уровне понятия)

интеграции сложных объектно-ориентированных программ эта организация предлагает использовать взаимосвязанную иерархию мета-моделей и моделей из четырех уровней. Эта иерархия носит название Meta Object Facility, MOF (Средства Мета-Объектов).

Уровни MOF определяются следующим образом, начиная с самого конкретного:

- ♦ уровень пользовательской информации состоит из той информации о реальном мире, которую требуется описать и непосредственно использовать в работе программы. Эту информацию обычно называют данными;

- ♦ уровень модели состоит из мета-данных, которые описывают структуру и связи информации. Именно мета-данные чаще всего в информатике называют моделями;

<sup>5</sup> OMG — Object Management Group. Международная организация, занимающаяся широким кругом вопросов эффективного моделирования и разработки сложных объектно-ориентированных систем. OMG является родоначальником языка моделирования UML и основанных на нем методов проектирования программного обеспечения.

◆ уровень мета-модели состоит из описаний структуры и мысла (семантики) мета-данных (моделей). То есть на этом уровне располагаются мета-мета-данные, или как их чаще называют — мета-модель. Мета-модель можно считать единым языком, на котором описываются различные виды данных;

◆ уровень мета-мета-модели состоит из описаний структуры и смысла мета-мета-данных. Иными словами — это единый язык, на котором описываются различные виды мета-моделей.

На рис. 1.24 показан пример одной из возможных реализаций каждого из четырех уровней иерархии MOF. В этом конкретном примере на уровне мета-модели используется язык для представления простых, не связанных между собой записей — т.е. определяется составной тип Record (например, записи о работниках).

Содержимое уровней определяется следующим образом:

◆ *уровень пользовательской информации* содержит конкретные экземпляры записей с информацией об определенных сотрудниках;

◆ *уровень модели* включает мета-данные, которые описывают внутреннюю структуру типа данных Record с именем Employee для хранения информации о сотруднике. Этот тип данных имеет два поля с заданными именами и типами;

◆ *уровень мета-модели* определяет, что собственно означает «являться типом данных Record». Это определение выражается через содержимое мета-класса с именем «Record». Он содержит два мета-атрибута: первый определяет имя типа Record, а второй — атрибуты. Таким же образом особый мета-класс определяет смысл элемента Field;

◆ *уровень мета-мета-модели* обычно жестко задан заранее и выявляет все те конструкции, которые используются для определения мета-моделей. Так, в нашем случае на этом уровне определяются такие понятия, как мета-класс, мета-атрибут и т.п.

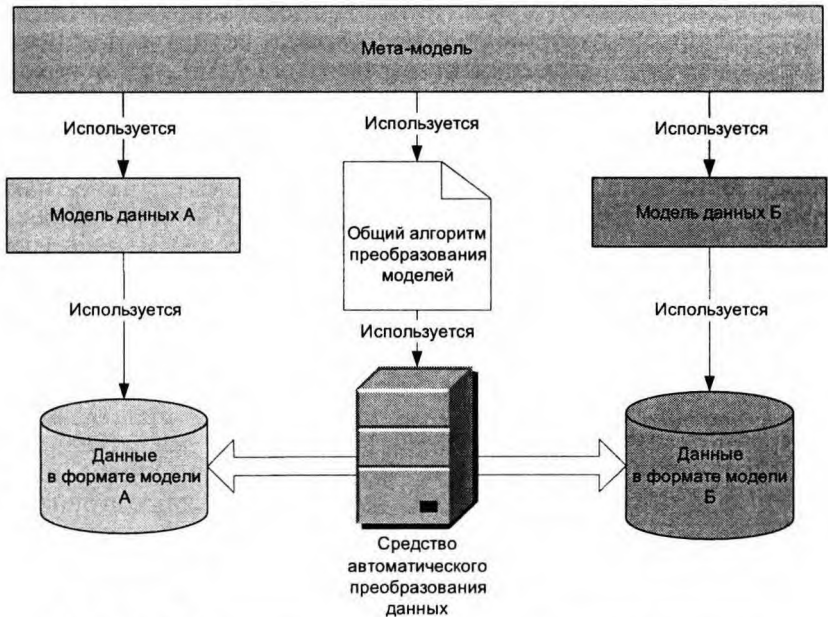
Подход на основе мета-моделирования предоставляет разработчику расширяемый механизм для описания всех деталей архитектуры программной системы. Он позволяет повторно использовать существующие возможности (пользовательский интерфейс, управление жизненным циклом

объектов и т.п.) и при этом обеспечивает модульную интеграцию с чужими системами и структурами данных. С использованием механизма мета-моделей интеграция структур данных становится возможной за счет того, что само описание этих структур доступно в программе в виде данных, с которыми можно производить типичные операции (чтение, модификация, создание, удаление и т.п.). Так, например, в программе становится возможным узнать, какую внутреннюю структуру имеет тот или иной, заранее неизвестный тип данных, какая используется кодировка данных и т.п. Поэтому во время работы программы можно динамически «расшифровать» структуру экземпляра любого типа данных и преобразовать его значения в другой тип по потребности алгоритма или пользователя<sup>6</sup>. Возможность одновременного манипулирования структурой данных и мета-данных на одном уровне позволяет реализовывать общие алгоритмы, которые не зависят от конкретных моделей или обладают возможностью единообразной работы с элементами различных моделей, представленных с помощью одной и той же мета-модели. Например, становится возможным создавать общие алгоритмы автоматизированного преобразования типов данных одной модели в типы данных другой модели. Эти возможности особенно полезны для решения задач автоматической генерации исходного кода по моделям, автоматизированного манипулирования разнородными данными, а также динамической сборки программных систем на основе независимых программных компонент (рис. 1.25).

Нужно отметить, что OMG предложила не только теоретическую концепцию, но и практические средства реализации мета-моделирования. OMG предлагает для создания мета-моделей и моделей использовать один и тот же язык моделирования — язык UML. В этом случае мета-модель определяет модель языка UML на самом высоком уровне абстракции и является наиболее компактным ее описанием — все основные понятия языка UML (класс, атрибут, операция, компонент, ассоциация и т.д.), их атри-

---

<sup>6</sup> Это остается верным до тех пор, пока используются типы данных, определенные с использованием иерархии MOF.



**Рис. 1.25.** Принципы автоматизированного преобразования разнородных данных на основе мета-моделирования

буты и взаимосвязи формально определяются на уровне мета-модели. Мета-модель языка UML имеет довольно сложную структуру, которая включает в себя порядка 90 мета-классов, более 100 мета-ассоциаций, число которых возрастает с появлением новых версий языка. Чтобы справиться с этой сложностью языка UML, все его элементы организованы в логические пакеты. Поэтому рассмотрение языка UML на метамодельном уровне заключается в описании трех его наиболее общих логических блоков или пакетов: основные элементы, элементы поведения и общие механизмы.

Модель в контексте языка UML является экземпляром мета-модели в том смысле, что любая конкретная модель системы на языке UML должна использовать только понятия мета-модели, конкретизировав их применительно к

данной ситуации. Это уровень для описания информации о конкретной предметной области. Однако если для построения модели используются понятия языка UML, то необходима полная согласованность понятий уровня модели с базовыми понятиями языка UML уровня мета-модели. Примерами понятий уровня модели могут служить, например, имена полей проектируемой базы данных, такие, как имя и фамилия сотрудника, возраст, должность, адрес, телефон. При этом данные понятия используются лишь как имена соответствующих информационных атрибутов.

Конкретизация понятий модели происходит на уровне объектов. В настоящем контексте объект является экземпляром модели, поскольку содержит конкретную информацию относительно того, чему в действительности соответствуют те или иные понятия модели.



## Глава 2

---

### Редактор спецификации Workflow-процессов

В предыдущей главе указывалось на преимущества процессно-ориентированного метода управления организацией. Однако само по себе решение описать существующие (или создать новые) бизнес-процессы предприятия, а затем следовать им в повседневной практике недостаточно для повышения эффективности управления — ведь создание бизнес-процесса во многом схоже с написанием должностных инструкций, регламентов и т.п. А мы из собственного опыта знаем, что зачастую реальная деятельность предприятия далека от правил, изложенных в таких инструкциях. Для того, чтобы разработка бизнес-процессов не была таким же безрезультатным занятием, как сочинение инструкций, необходимо иметь набор инструментов, с помощью которых можно осуществлять гарантированно точное выполнение и контроль за бизнес-процессами во всех подразделениях организации.

На сегодняшний день набором таких инструментов можно считать практически реализации программных систем в соответствии с рекомендациями консорциума WfMC. Мы уже познакомились с основой для их широкого использования — языком описания потоков работ (workflow) — XPDЛ. Теперь можно рассмотреть конкретные примеры программ, применяемых для создания описаний на языке XPDЛ и автоматизированного выполнения этих описаний. В нашем курсе будут изучаться две программы:

- ◆ редактор спецификаций workflow-процессов JaWE;
- ◆ распорядитель выполнения работ (workflow Engine) Shark.

Оба этих программных продукта являются свободно распространяемыми программами на языке Java и совмест-

но разрабатываются в рамках международного консорциума ObjectWeb (<http://www.objectweb.org>). Этот консорциум объединяет разработчиков и исследователей из разных стран с целью создания эффективных программных технологий и программных продуктов различных категорий: интеграция различных приложений предприятия, электронная коммерция, распределенные вычислительные и информационные системы и т.п. Благодаря использованию стандартных объектно-ориентированных решений и унификации программной архитектуры обеспечивается тесная интеграция JaWE и Shark между собой, а также с другими проектами консорциума ObjectWeb. Это позволяет на основе различных свободно распространяемых проектов строить законченные решения корпоративных информационных систем для различных предприятий.

## **2.1. Редактор спецификаций JaWE (Java Workflow Editor)<sup>1</sup>**

JaWE — это визуальное средство для создания бизнес-процессов, которое обеспечивает выполнение трех основных процедур:

- ◆ построение определения процесса в наглядной графической форме;
- ◆ импортирование определений из внешних файлов;
- ◆ экспортирование корректных определений во внешние текстовые и графические файлы.

Простота JaWE позволяет пользователю быстро создавать определения workflow-процессов, проверять и сохранять их для дальнейшего использования другими системами. Заключительным результатом моделирования бизнес-процесса в редакторе JaWE является текстовый файл в формате XPDL, который может интерпретироваться во время выполнения workflow различными распорядителями работ (workflow engines), в частности продуктом Shark. За счет использования в JaWE стандартной мета-модели, разработанной консорциу-

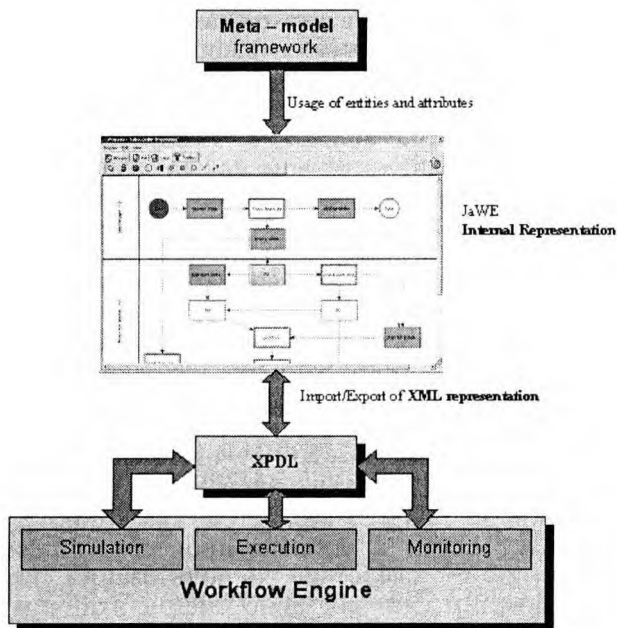
---

<sup>1</sup> Официальный веб-сайт программы JaWE, откуда можно получить справочную информацию и саму программу, находится по адресу: <http://jawe.objectweb.org>. Данная глава основана на использовании редактора JaWE версии 1.4-1.

мом WfMC, определение workflow-процесса, сгенерированное JaWE, является пригодным к обмену и использованию в различных системах Workflow Management (рис. 2.1).

Из предыдущих разделов понятно, что мета-модель идентифицирует все разрешенные к использованию объекты для определения бизнес-процесса, их атрибуты и связи между объектами. Имеется обязательный минимальный набор объектов, которые должны поддерживаться любой системой, претендующей на понимание языка XPDL. В него входят такие объекты:

- ◆ определение процесса (Workflow process Definition);
- ◆ функции процесса (Workflow Process Activity);
- ◆ переходы процесса (Transition Information);
- ◆ объявление участника (Workflow Participant Declaration);



**Рис. 2.1.** Следствие использования мета-модели XPDL в редакторе JaWE

- ◆ объявление приложения (Workflow Application Declaration);
- ◆ данные процесса (Workflow Relevant Data);
- ◆ данные системы и среды окружения (System & Environmental Data);
- ◆ архив ресурсов или организационная модель (Resource Repository or Organizational Model);
- ◆ типы данных и выражения (Data Types and Expressions).

Эта «минимальная мета-модель» идентифицирует стандартные используемые объекты в пределах определения процесса и описывает их значение. У каждого из вышеупомянутых объектов имеется стандартный набор атрибутов (для некоторых обязательный, для других необязательный), которые описывают характеристики объекта. Расширяемость обеспечивается за счет того, что разрешается к определению каждого объекта добавлять дополнительные атрибуты (так называемые «расширенные атрибуты», «extended attributes»).

В JaWE поддерживается работа со всеми объектами, определенными в мета-модели, за исключением «Данных системы и среды окружения», а также «Архива ресурсов». «Данные системы и среды окружения» используются системой Workflow Management во время выполнения процесса, поэтому их поддержка просто не должна входить в обязанности редактора JaWE. Что касается архива ресурсов, то в редакторе JaWE при разработке собственного бизнес-процесса можно сослаться на любое определение участников, процессов, приложений и данных, находящееся во внешнем XPDЛ-файле. Он может содержать сложную организационную модель, большое количество совместно используемых приложений или данных и, таким образом, являться полным аналогом архива ресурсов.

Логически в JaWE выделяется два уровня: уровень пакета (Package Level) и уровень процесса (Process Level). Это разделение — следствие использования концепции пакетов. Напомним, что согласно спецификациям консорциума WfMC пакет (Package) является контейнером, где сгруппированы вместе несколько процессов и связанные данные, используемые этими процессами совместно. В

пределах пакета — эти объекты могут использоваться во всех определениях других процессов, содержащихся в пакете. Такими объектами с глобальной областью видимости являются:

- ◆ определение процесса;
- ◆ спецификация участника;
- ◆ объявление приложения;
- ◆ определение данных процесса.

Уровень процесса управляет объектами и атрибутами в пределах определения одного workflow-процесса. На уровне процесса некоторые данные, впервые определенные на уровне пакета, могут быть переопределены заново.

Обычно принято создавать отдельный пакет для каждого самостоятельного бизнес-процесса. В этом пакете должны содержаться все необходимые определения вспомогательных подпроцессов, объявления связанных с ними инструментальных средств (приложения), определение участников и т.п. Также возможно в один пакет поместить только часть полного определения процесса (группу подпроцессов) или определения, совместно используемые несколькими процессами (например, определения участников или объявления приложений). Затем на такой пакет можно ссылаться из других пакетов.

## ***2.2. Уровень пакета (Package Level)***

На уровень пакета JaWE переходит сразу после запуска. При этом графический интерфейс имеет вид, представленный на рис. 2.2. и 2.3.

Графический интерфейс на уровне пакета разделен на несколько областей. В левой области основного окна находится дерево всех использующихся пакетов. Корень дерева — это текущий пакет, с которым идет работа, а остальные вершины дерева — импортированные внешние пакеты. Если, в свою очередь, эти пакеты ссылаются на другие внешние пакеты, то они также представляются в виде подчиненных вершин в дереве. Листья дерева (т.е. вершины, не имеющие подчиненных вершин) обозначают такие пакеты, которые не имеют ссылок на внешние пакеты, либо ссылаются на такие внешние пакеты, которые уже

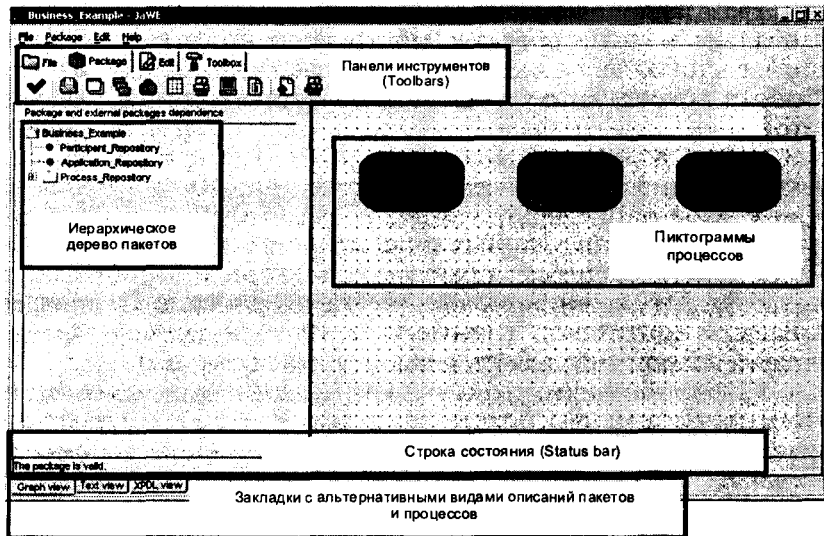


Рис. 2.2. Интерфейс на уровне пакета (Package Level)

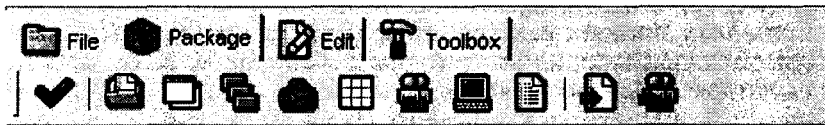


Рис. 2.3. Инструментальная панель уровня пакета

были включены в дерево раньше. Так как все пакеты имеют уникальный ID (идентификатор), то в дереве допускаются циклические зависимости.

В правой области основного окна в виде прямоугольников с закругленными углами отображаются все бизнес-процессы, которые определены в пакете, выбранном на данный момент в дереве пакетов. На уровне пакета большая часть информации о процессе скрыта. Получить доступ к этой информации можно через команду «Properties» контекстного меню процесса (контекстное меню вызывается по нажатию правой кнопки мыши, в то время как курсор находится на прямоугольнике процесса). Нужно сказать, что на уровне пакета можно выбрать различный

вид представления информации на языке XPDЛ. По умолчанию установлен графический вид (graphical). Изменить вид на текстовый (text) или на вид в форме XML-документа (xpdл) можно, выбрав соответствующую закладку в нижней части окна редактора. Другие области на уровне пакета следующие: заголовок, меню, панели инструментов, строка состояния.

Каждый раз, когда пользователь хочет открыть новый документ (для этого выбирается команда основного меню «File/New»), требуется создание нового пакета. Об этом пользователя предупреждает сообщение. После закрытия этого окна появляется служебный диалог, в котором можно задать все необходимые параметры вновь создаваемого пакета (рис. 2.4). Описание основных элементов этого окна и их назначение вы найдете в оригинальной документации.

Package " - properties

General

Id:

Name:

Graph conformance:

Script

Type:

Version:

Grammar:

Extended attributes

Name	Value
------	-------

↑ New

↻ Edit

↓ Delete

OK

Рис. 2.4. Основные параметры пакета

Как только установлены основные параметры пакета, появляется возможность продолжить работу по определению различных объектов, которые он будет содержать (приложения, участники, процессы и т.п.). При этом в любой момент работы на уровне пакета можно изменить его ранее установленные параметры, выбрав пункт основного меню «Package/Properties».

Добавление новых процессов в состав текущего пакета происходит с помощью инструмента «Insert Process» на панели инструментов «Toolbox» (рис. 2.5).

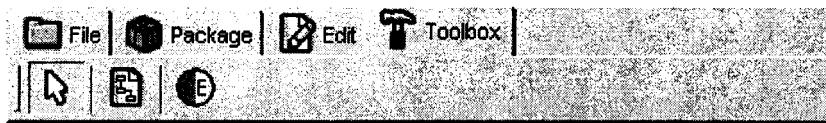


Рис. 2.5. Панель инструментов «Toolbox» на уровне пакета

Для того, чтобы начать определение добавленного процесса, необходимо переместить на пиктограмму процесса курсор мыши, нажать ее правую клавишу, вызвав на экран контекстное меню, а из контекстного меню вызвать команду «Edit». При этом происходит переход на логический уровень процесса и открывается новое окно для редактирования бизнес-процесса. Кроме того, двойной щелчок на символе любого процесса также переводит редактор JaWE на уровень процесса. Отметим, что процессы, импортируемые из внешних пакетов, недоступны для редактирования — они видимы только как ссылки.

### 2.3. Уровень процесса (*Process level*)

На этом логическом уровне основной рабочей частью редактора JaWE является окно для редактирования бизнес-процесса (рис. 2.6.). Оно используется для графического (или текстового) представления отдельных элементов, составляющих процесс (т.е. функций и переходов), а также для определения атрибутов этих элементов. Кроме того, на этом уровне имеется возможность переопределения различных вспомогательных элементов: объявлений приложений, определений участников процесса, данных процесса.



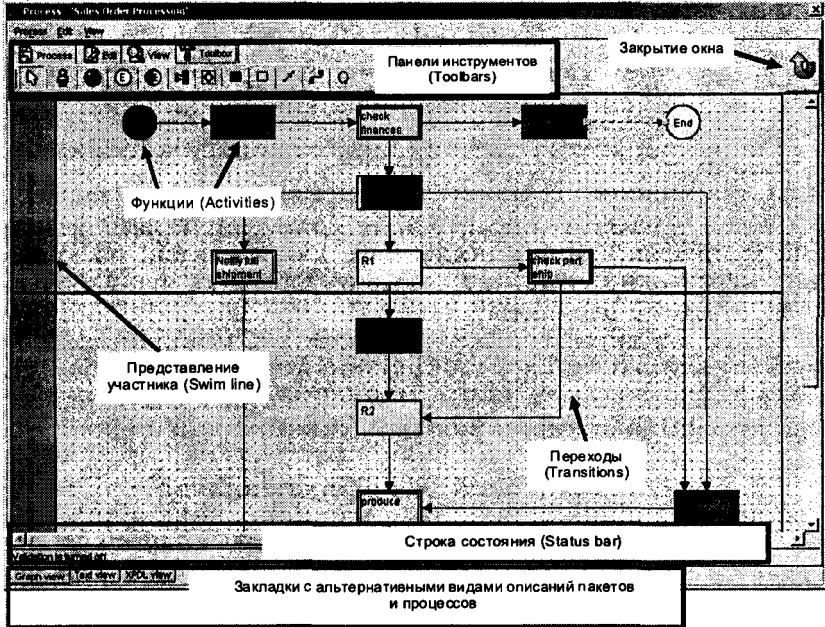


Рис. 2.6. Интерфейс на уровне процесса

Для того, чтобы возвратиться обратно на уровень пакета и закрыть окно с определением процесса, необходимо нажать на кнопку с голубой стрелкой в верхнем правом углу окна. Если выбрано графическое представление, то основную часть окна редактирования бизнес-процесса занимает рабочая область. В ней наглядно представляются функции и переходы, а также показано, какие функции выполняет каждый из участников. Для этого каждому участнику соответствует своя горизонтальная полоса (обычно такие полосы называют «swim line»). В самой левой части этой полосы, в цветном прямоугольнике, указывается идентификатор участника. Участники, функции и другие элементы процесса добавляются на рабочую область с использованием кнопок-инструментов, находящихся на линейке инструментов «Toolbox» (рис. 2.7).

Нужно отметить, что в редакторе JaWE каждая функция процесса изначально должна быть связана с определен-

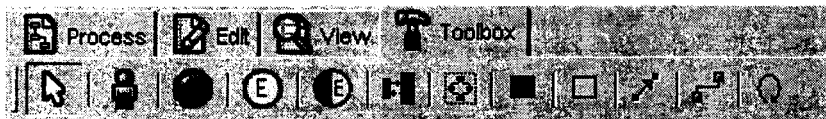


Рис. 2.7. Панель инструментов «Toolbox» на уровне процесса

ным участником. Поэтому при создании нового бизнес-процесса первое, что делается, — создается первый участник (participant). Лишь после того, как добавлен, по крайней мере, один участник, можно добавлять на рабочую область другие элементы.

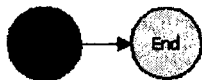
При создании функций пользователь может использовать несколько различных их типов в соответствии со спецификацией языка XPDЛ. Ниже следует краткое описание разрешенных типов функций в редакторе JaWE.

### 2.3.1. Функция общего вида (Generic Activity)

Функции общего вида соответствует кнопка ■ на панели инструментов «Toolbox». При выборе этого инструмента курсор меняет свою форму, указывая на тип выбранной функции. Двойное нажатие левой клавиши мыши на кнопке инструмента активизирует так называемый «постоянный режим» (persistent mode). В этом режиме инструмент остается активным после размещения очередной функции на рабочей области. Выключить постоянный режим можно, выбрав другой инструмент на панели. Объекты создаются на рабочей области с использованием параметров по умолчанию (например, таким параметром является размер пиктограммы). Параметры по умолчанию могут быть изменены пользователем во время работы.


#### *Начало и конец процесса (Start-End)*

Началу процесса соответствует инструмент ●, а концу процесса — (E). На рабочей области пиктограммы соответственно выглядят так:




Несмотря на то, что спецификация XPDЛ явно не определяет начала и конца процесса, они, безусловно, нужны для указания места, где процесс начинается и заканчивается. Обратите внимание на то, что согласно правилам редактора JaWE начало и конец процесса могут иметь ровно один исходящий/входящий переход.




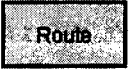
### Подпроцесс (Subflow)

Подпроцесс  является функцией, в которой реализуется другой бизнес-процесс. Поэтому при выполнении такой функции данные текущего процесса и приложения, в нем определенные, большой роли не играют. Важное значение имеют режим выполнения подпроцесса (execution mode: синхронный или асинхронный), а также фактические параметры, которые передаются в подпроцесс. Другие свойства подпроцесса аналогичны функциям общего вида.




### Функция-группа (Block Activity)

Функция-группа  исполняет набор функций (ActivitySet). Исполнение начинается с первой функции набора и продолжается до достижения заключительной функции в наборе (функция, не имеющая выходных переходов). Затем происходит маршрутизация по правилам, определенным выходными переходами самой функции-группы.

В редакторе JaWE используется следующая графическая нотация для обозначения различных типов функций:

	Функция общего вида Generic activity
	Группа Block activity
	Подпроцесс Sub-flow activity
	Маршрутизация Router Не выполняет никаких действий, служит для синхронизации параллельно выполняемых функций

Переходы (Transition) связывают две функции. В редакторе используется три вида переходов — простые переходы (simple), переходы с самостоятельной маршрутизацией (self-routed) и циклические переходы (circular).

Переходы можно создать, используя инструменты  (simple),  (self-routed) или  (circular). Для более красивого представления бизнес-процесса можно использовать breakpoint на указателе перехода.

### 2.3.2. Текстовое представление (Text view)

Описание бизнес-процесса можно увидеть в текстовом представлении (рис. 2.8). Оно используется для представления структуры дерева XPDL-документа. Также в нем можно быстро редактировать XPDL-документ.

### 2.3.3. XPDL-представление (XPDL View)

XPDL-представление используется для показа структуры разработанного определения процесса в формате XML (или, более точно, в XPDL). Каждый объект или атрибут, который графически добавлен или изменен в JaWE, автоматически отображается в панели — XPDL view (рис. 2.9).

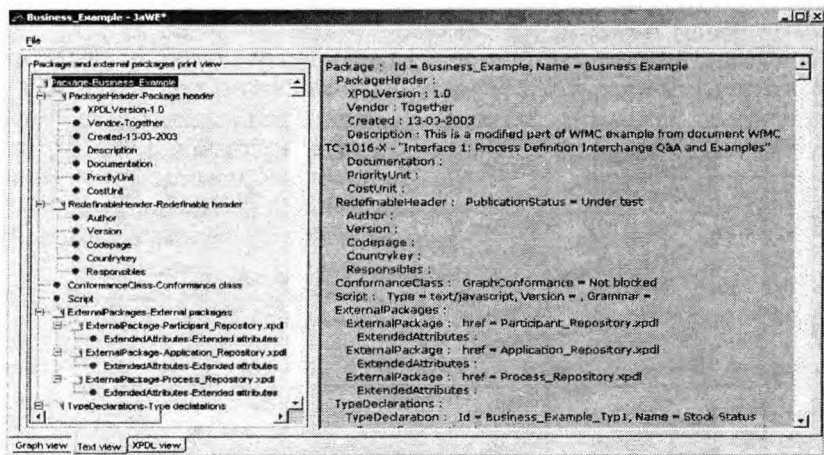


Рис. 2.8. Текстовое представление — Text view

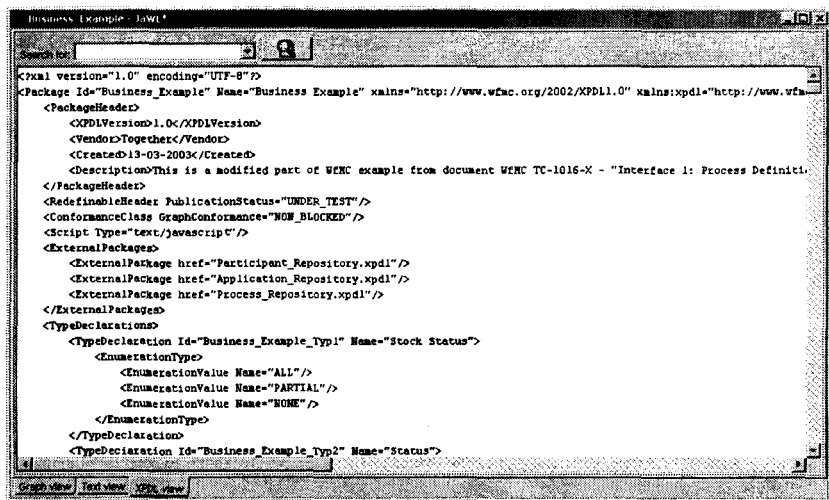


Рис. 2.9. XPD представление — XPD view

### 2.3.4. Проверка правильности XPD-документа

Когда пользователь открывает или записывает XPD-файл с определением бизнес-процессов, то редактор JaWE автоматически производит серию проверок:

- ◆ проверка на соответствие стандартному синтаксису языка XPD (соответствие XML-схеме);
- ◆ проверка на правильность соединения функций между собой переходами;
- ◆ проверка на соответствие графа бизнес-процесса установленному классу соответствия (conformance class);
- ◆ проверка на отсутствие логических противоречий.

Если в результате проверок обнаружены ошибки, то появляется диалоговое окно, содержащее описание ошибки и указание на место в файле, где она обнаружена. При этом пользователю предоставляется выбор: открыть документ, не смотря на ошибки, или отказаться от загрузки файла в редактор. Пример отчета по результатам проверки показан на рис. 2.10.

Выполнить проверку текущего бизнес-процесса можно и во время работы. Для этого служит команда основного меню «Process/Check Validity».

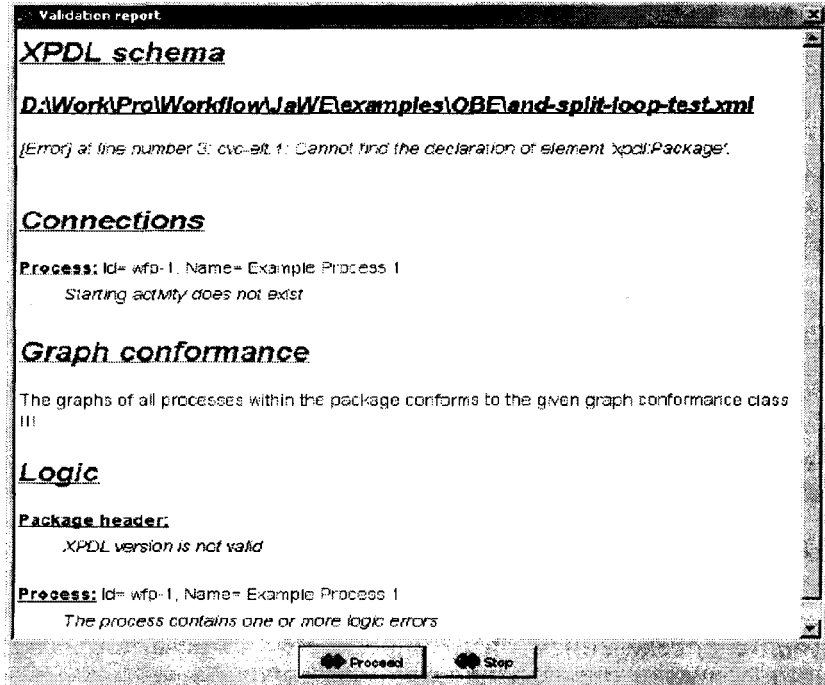



Рис. 2.10. Окно отчета по результатам проверки правильности описания процесса

### 2.3.5. Предопределенные комбинации клавиш в редакторе JaWE

Любой объект в редакторе JaWE можно выделить с использованием мыши и инструмента выделения . Кроме того, можно перемещаться по объектам с использованием клавиатуры. Для этого в редакторе определены следующие комбинации клавиш для перемещения по графу пакета или процесса:

◆ ALT+ENTER — отображаются параметры текущего выделенного объекта;

◆ ALT+CURSOR\_KEY — перемещение по объектам графа;

- ◆ ALT+BACKSPACE — закрыть окно с графом процесса;
- ◆ F2 — отображаются свойства текущей сущности графа;
- ◆ CTRL-Z — отмена последнего выполненного действия.

## 2.4. Создание бизнес-процесса в среде разработки JAWE

В этом разделе будет определена последовательность шагов, необходимых для описания бизнес-процесса «обработка заказа», введенного в гл. 1. С целью облегчения описания процесс обработки заказа в данной главе несколько упрощен по сравнению с первоначальным вариантом (рис. 2.11, 2.12).

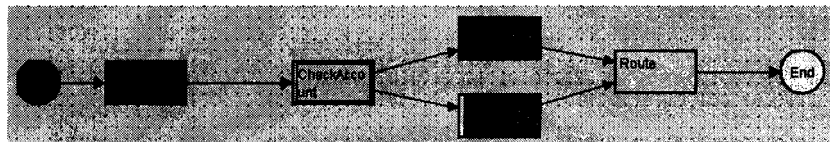


Рис. 2.11. Графическое представление упрощенного варианта бизнес-процесса «обработка заказа»

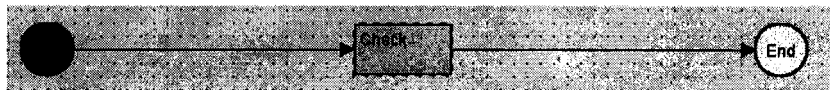


Рис. 2.12. Графическое представление упрощенного варианта подпроцесса «проверка платежеспособности» («CheckAccount»)

### 2.4.1. Шаг 0. Установка продукта

Инсталлятор редактора JaWE может быть скачан бесплатно с сайта консорциума ObjectWeb:



[http://forge.objectweb.org/project/showfiles.php?group\\_id=50](http://forge.objectweb.org/project/showfiles.php?group_id=50)).

Существуют версии для различных операционных систем.



Программа предъявляет следующие требования к системе (для версии JaWE 1.4): JDK 1.4.1 или старше.

## 2.4.2. Шаг 1. Начало работы

После инсталляции и запуска JaWE открывается рабочая область с отображением уровня пакета (рис. 2.13).

На панели инструментов выбираем инструмент «Select»  и выделяем на дереве пакетов пакет верхнего уровня (серая точка). После чего становится возможен выбор других инструментальных средств  для уровня пакета.

## 2.4.3. Шаг 2. Создание процесса

На панели инструментов выбираем инструмент «Insert Process»  и вставляем процесс на уровень процессов (правая область). После вставки нового процесса в правой области появляется пиктограмма процесса — .

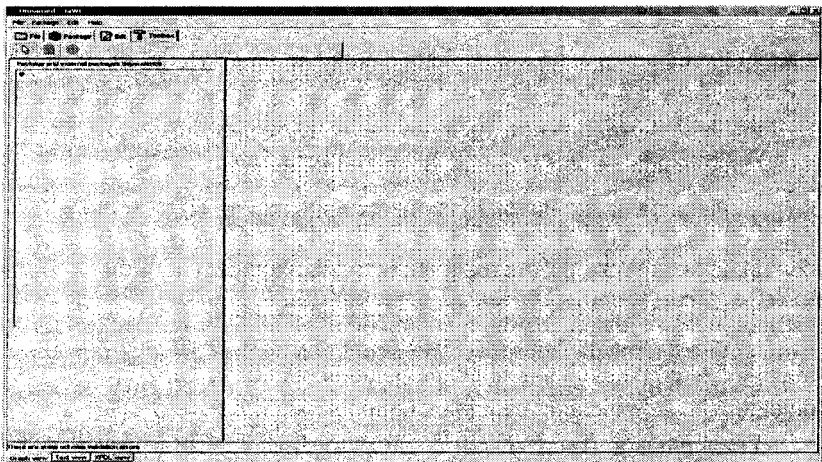


Рис. 2.13. Первоначальный вид интерфейса JaWE после запуска



Теперь надо определить содержимое и параметры созданного процесса. Устанавливаем курсор мыши на пиктограмме процесса и щелкаем правой кнопкой — появляется контекстное меню, в нем выбираем пункт «Edit». Открывается окно графического представления бизнес-процесса (рис. 2.14).

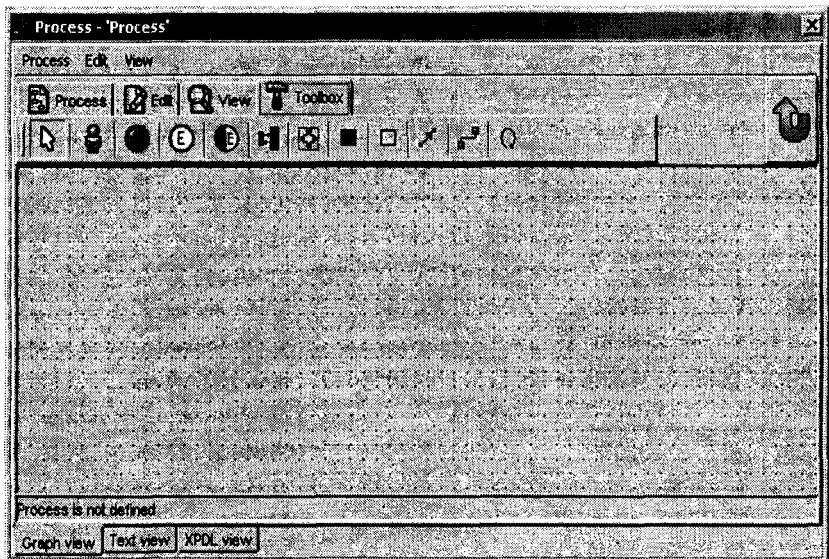


Рис. 2.14. Вид интерфейса JaWE на уровне процесса

В нем на панели инструментов «Toolbox» доступны для работы инструменты, перечисленные на стр. 86.

Согласно принятым правилам при создании процесса первое, что надо сделать — это создать участника процесса. Выбираем инструмент «Participant» из панели инструментов «Toolbox» и перетаскиваем пиктограмму на рабочую область окна. После нажатия левой клавиши мыши на рабочей области окна появляется диалог для определения параметров участника (рис. 2.15).

Поле «идентификатор» (Id) можно оставить без изменений. Поле «имя» (name) заполняется произвольно. Далее устанавливается один из разрешенных типов участника:

	Выбор (Select)	Выбор отдельных элементов бизнес-процесса
	Участник (Participant)	Добавление участника на уровне процессов. Это может быть только участник без действий
	Начало (start)	Начало процесса
	Завершение (End)	Завершение процесса
	Вставка начала и завершения (Insert start/end bubbles)	Автоматическая установка начала и завершения
	Подпроцесс (Subflow activity)	Вставка подпроцесса
	Группа (Block activity)	Вставка функции-группы
	Функция общего вида (Generic activity)	Вставка функции общего вида
	Маршрутизатор (Route activity)	Вставка функции-маршрутизатора
	Переход (Transition)	Вставка перехода между двумя функциями в форме отрезка прямой.
	Переход (Transition)	Вставка перехода между двумя функциями в форме ломаной
	Переход (Transition)	Вставка циклического перехода

«набор ресурсов» (resource set), «ресурс» (resource), «роль» (role), «подразделение организации» (organizational unit), «человек» (human), «система» (system). Также на этом уровне можно добавить комментарий (description), определив, какую функциональную нагрузку несет этот участник.

После определения участника можно закрыть диалог и выполнить следующий шаг — добавить на рабочую область окна начало и конец процесса («Start» и «End»). Запомните, что выполняться эти функции обязательно должны одним участником.

**New participant?**

Participant

Id: 1\_Par1

Name:

Type

Resource set

Role

Human

Resource

Organizational unit

System

Description:

External reference

Xref:

Location: ...

Namespace: ...

Extended attributes

Name	Value
------	-------

↑

↓

New

Edit

Delete

OK

Cancel

Рис. 2.15. Диалог для определения параметров участника процесса

### 2.4.4. Шаг 3. Запись в файл и создание пакета

Перед тем, как продолжить работу по определению бизнес-процесса, стоит сохранить сделанную работу во внешнем файле в формате XPDЛ. Для этого временно закрываем окно определения процесса (голубая стрелка в правом верхнем углу окна), переходя на уровень пакета, и вызываем из основного меню команду «File/Save». При этом редактор создает новый пакет, в котором хранится описание текущего бизнес-процесса. Вам потребуется установить параметры этого пакета в диалоговом окне (рис. 2.16):

♦ поле «идентификатор» (Id) — самое важное поле, далее оно будет только для чтения, можно ввести произвольное обозначение;

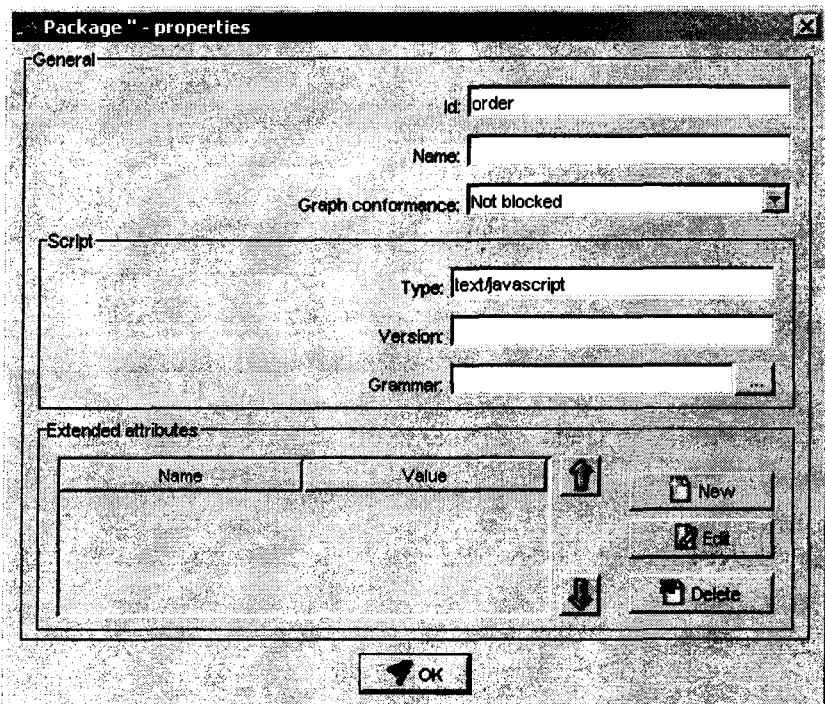


Рис. 2.16. Диалог для определения параметров пакета

◆ поле «имя» (Name) — подробное название вашего пакета.

Хотя все остальные параметры можно оставить без изменения, обратите внимание на группу параметров «Script». Здесь определяются синтаксические правила, которым должны соответствовать логические выражения, используемые в функциях — маршрутизаторах и переходах.

Имя	Обязателен?	Описание
Type	ДА	Определяет конкретный скриптовый язык, используемый для определения условий переходов между функциями бизнес-процесса. Рекомендуется выбирать тип скриптового языка из следующего списка: text/javascript, text/vbscript, text/tcl, text/ecmascript, text/xml.
Version	НЕТ	Версия скриптового языка.
Grammar	НЕТ	Является ссылкой на внешний документ, в котором формально определяются синтаксис и грамматика скриптового языка. Таким файлом, например, является XML-схема, файл DTD или набор правил Бекуса-Науэра.

После ввода основных параметров создается новый пакет и все его содержимое (описание бизнес-процесса, участников, приложения и т.д.) записывается в текстовый файл формата XPDL<sup>8</sup>.

Напомним, что для более подробного описания и (или) добавления каких-либо дополнительных параметров (например, внешних ссылок) в определение пакета можно воспользоваться во время работы на уровне пакетов инст-

<sup>8</sup> На этапе записи, возможно, будет выдано предупреждение о нарушениях в логической структуре процесса. Его можно проигнорировать и продолжить операцию записи в файл.

рументом «Package Properties» из панели инструментов «Package» (рис. 2.17) или командой «Package/Properties» основного меню.

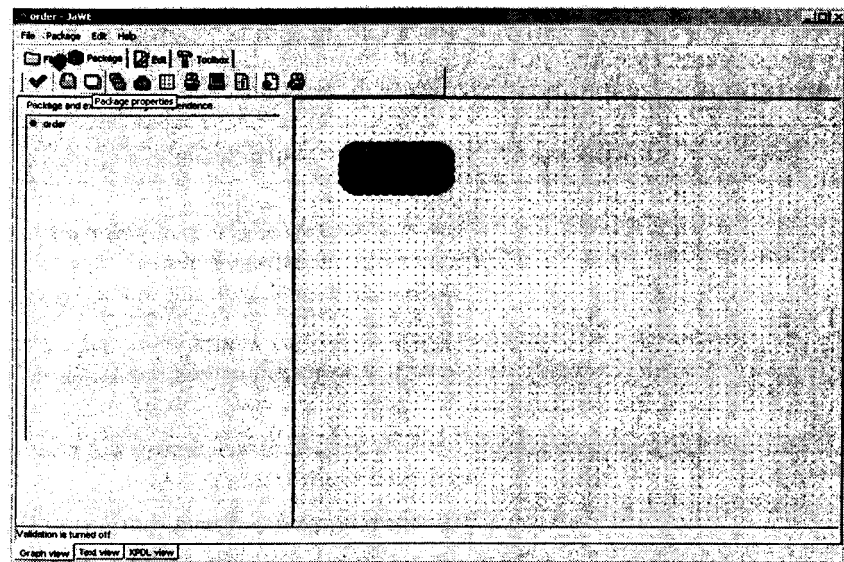


Рис. 2.17. Интерфейс на уровне пакета после создания процесса

#### 2.4.5. Шаг 4. Работа над бизнес-процессом: создание функции «Получение заказа» (Receive Order)

Теперь приступим к непосредственной реализации бизнес-процесса. Для этого опять переходим на уровень процессов, вызывая команду редактирования «Edit» из контекстного меню процесса. Для создания функции «Получение заказа» («Receive Order») можно использовать инструмент «Generic activity» из панели инструментов «ToolBox». Выделяем этот инструмент и перемещаем курсор мыши на «swim line» первого участника. После нажатия левой клавиши на экране появляется символическое обозначение функции (рис. 2.18).

Когда курсор стоит на пиктограмме вставленной функции, двойным нажатием левой клавиши мыши можно вызвать диалоговое окно для определения свойств (рис. 2.19).

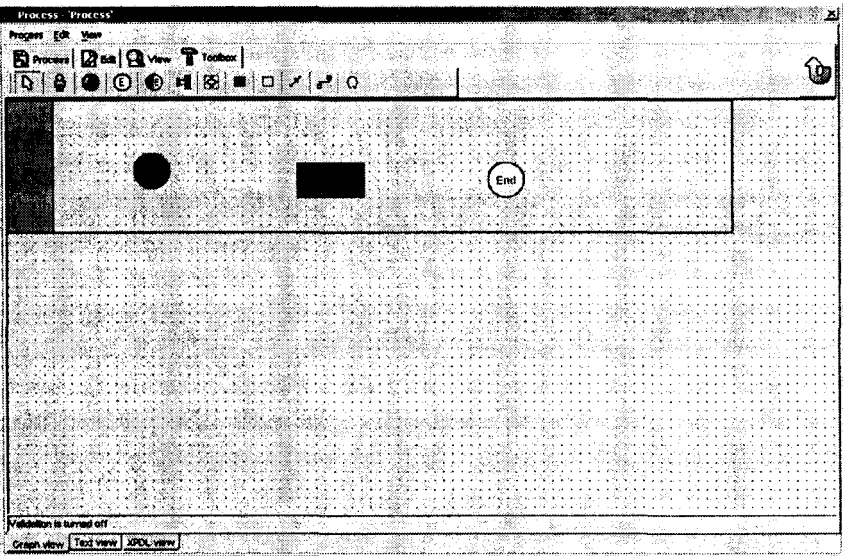


Рис. 2.18. Добавление функции

На закладке с общими свойствами (закладка «Generic») значение идентификатора генерируется автоматически. Поэтому единственное, что надо изменить, — это имя функции (параметр «Name»). Назовем эту функцию «Receive Order». Определение инструментов, с помощью которых выполняется функция (закладка «Tools»), ограничений на входящие переходы (закладка «Precondition»), ограничений на исходящие переходы (закладка «Postcondition»), а также определение параметров имитационных экспериментов (закладка «Simulation Information») будут выполнены позднее. Поэтому диалоговое окно можно закрыть.

#### *2.4.6. Шаг 5. Работа над бизнес-процессом: определение данных процесса и пользовательских типов данных*

В текущем бизнес-процессе нам потребуются следующие элементы данных (workflow relevant data), содержащие сведения о текущем обрабатываемом заказе:

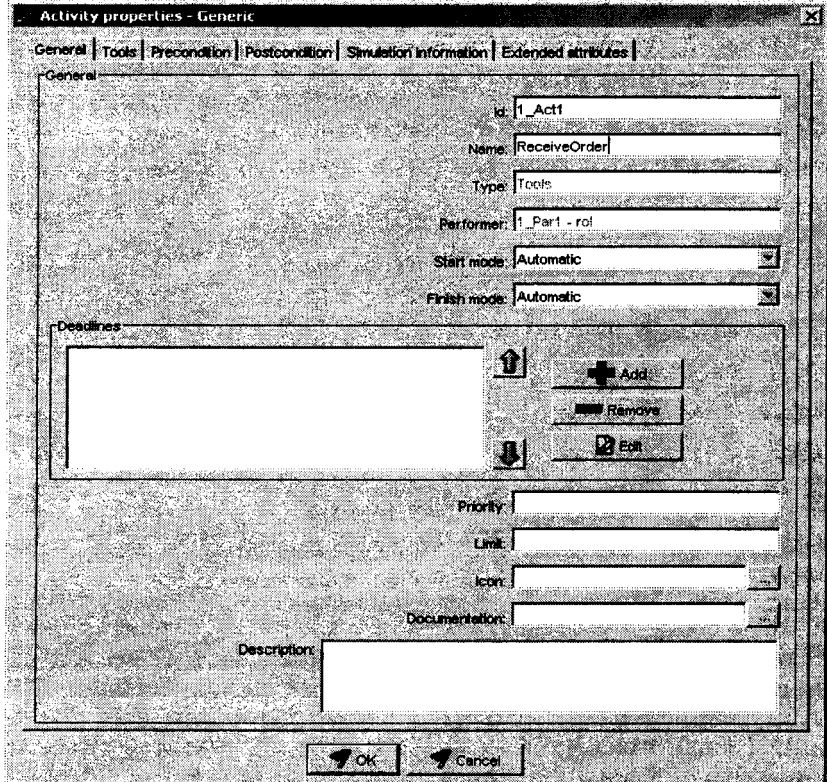


Рис. 2.19. Диалог для определения свойств функции

1. Customer Name (стандартный тип String) — содержит имя пользователя;
2. Order Number (стандартный тип Integer) — содержит номер заказа;
3. Account Number (стандартный тип String) — содержит номер счета пользователя;
4. Order Amount (стандартный тип Float) — содержит стоимость заказа;
5. Order Status (перечислимый тип со значениями {OK, NOT\_OK}) — состояние заказа.



Эти элементы будут использоваться в качестве входных или выходных данных при выполнении функций, а также в логических условиях, определяющих порядок срабатывания переходов. Покажем на примере определения элемента «Order Number», какие действия потребуется выполнить в редакторе JaWE.

Для определения данных, прежде всего, нужно активировать панель инструментов «Process» и выбрать на ней инструмент «Workflow relevant data». При этом появляется диалог «Process 'Process' – Workflow relevant data» (рис. 2.20).

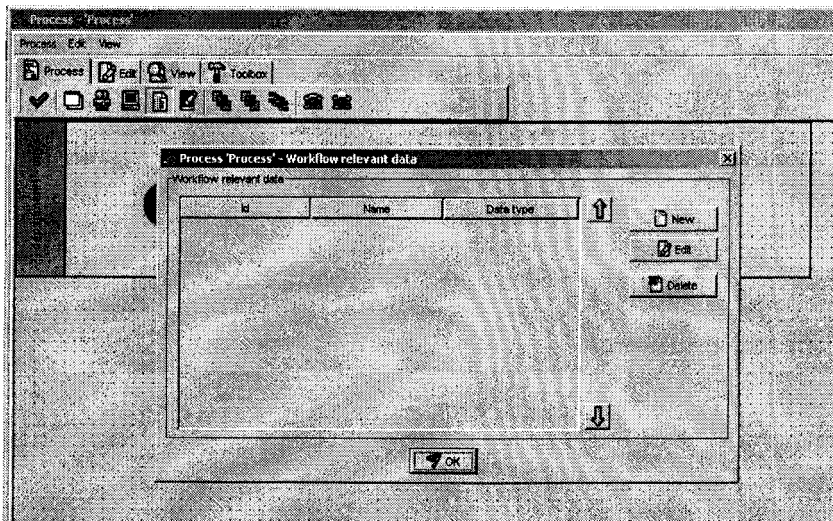


Рис. 2.20. Диалог с перечислением данных процесса

Добавление нового элемента данных производится путем нажатия кнопки «New» в этом диалоге. При этом появляется диалоговое окно «Workflow relevant data — defining» (рис. 2.21).

В этом окне вводим параметры нового элемента данных:

- ◆ Id: — Order\_Number (без пробелов!);
- ◆ Name: — Order Number (можно с пробелами);
- ◆ Type: — Basic Type (выбираем из списка значений);
- ◆ Sub-type: — Integer (выбираем из списка значений).

Workflow relevant data - defining

Workflow relevant data

Id:

Name:

Is array:

Type

Type:

Sub-type:

Initial value:

Length:

Description:

Extended attributes

Name	Value

**Рис. 2.21.** Диалог для определения нового элемента данных процесса

Все остальные параметры оставляем без изменения.

Элемент данных определен, теперь текущий диалог можно закрыть и возвратиться в диалог «Process 'Process' — Workflow relevant data» (рис. 2.22). Обратите внимание, что в нем появился только что определенный элемент данных.

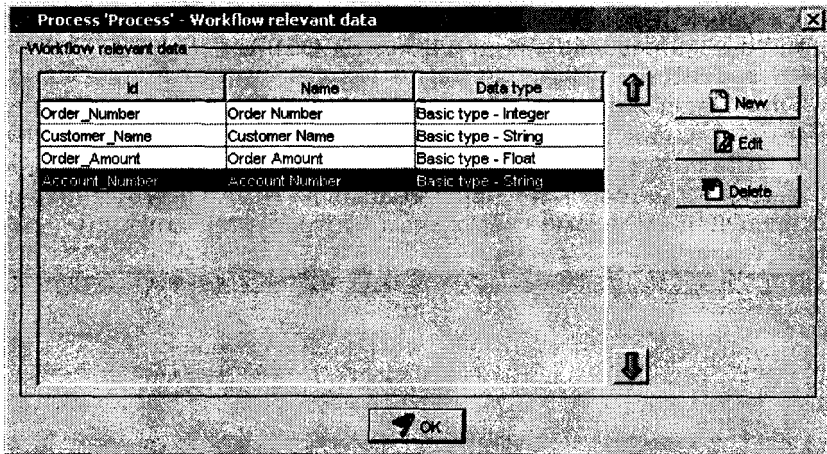


Рис. 2.22. Диалог с перечислением данных процесса после создания четырех элементов

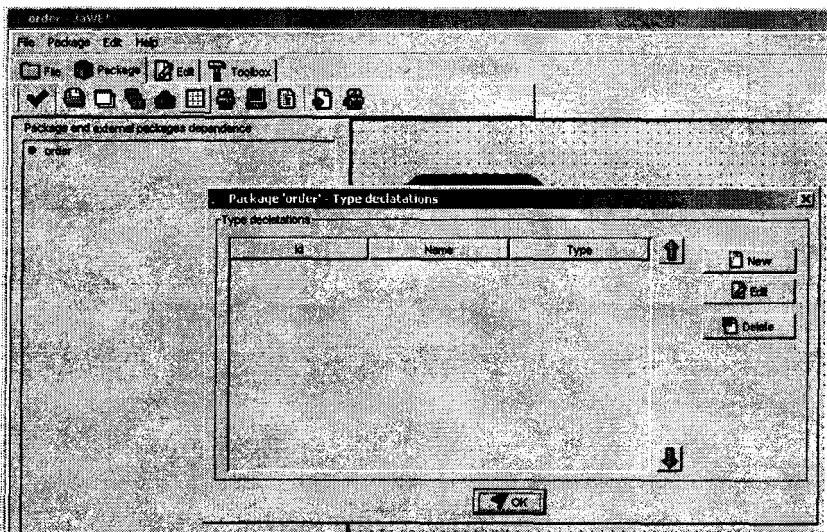


Рис. 2.23. Диалог с перечислением типов данных на уровне пакета

Повторяем процедуру определения элементов данных еще три раза, определяя такие элементы данных:

- ◆ Customer Name;
- ◆ Order Amount;
- ◆ Account Number.

Для пятого элемента данных — «Order Status» — выбран пользовательский перечислимый тип. Поэтому перед определением элемента данных потребуется этот тип со-

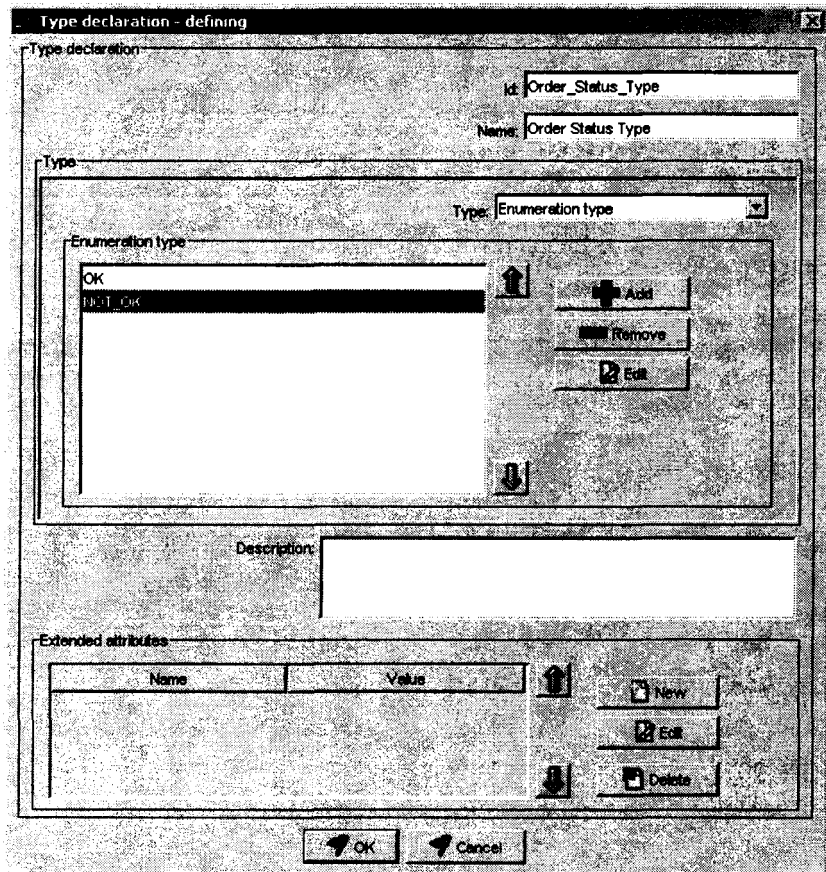


Рис. 2.24. Диалог для определения нового типа данных

здать. Лучше всего это сделать на уровне пакета. В таком случае новый тип данных будет доступен из различных процессов в текущем пакете. Для определения типа данных на уровне пакета закрываем все диалоги и окно определения процесса и переходим на уровень пакета. Из панели

The dialog box is titled "Workflow relevant data - defining". It contains the following fields and controls:

- Id:** Order\_Status
- Name:** Order Status
- Is array:** False
- Type:** Declared type
- Sub-type:** Order Status Type
- Initial value:** (empty field)
- Length:** (empty field)
- Description:** (empty text area)
- Extended attributes:** A table with two columns: Name and Value. The table is currently empty.
- Buttons:** Define, New, Edit, Delete, OK, Cancel.

Name	Value
------	-------

Рис. 2.25. Содержимое диалога для определения нового элемента данных в случае для элемента «Order\_Status»

инструментов «Package» выбираем инструмент «Type declarations». Появляется диалоговое окно «Package 'order' — type declaration», которое напоминает окно с перечислением элементов данных (рис. 2.23).

Так же, как и определение нового элемента данных, определение нового пользовательского типа данных производится нажатием кнопки «New». При этом появляется диалог «Type declaration — defining» (рис. 2.24).

Заполняем поля этого диалога следующим образом:

- ◆ Id: — Order\_Status\_Type (без пробелов!);
- ◆ Name: — Order Status Type (можно с пробелами);
- ◆ Type: — Enumeration Type (выбираем из списка значений).

После выбора значения для поля Type появляется рабочая область «Enumeration Type». В ней с помощью кнопки «Add» можно последовательно перечислить все разрешенные значения этого типа. Добавив два значения (OK и NOT\_OK), мы можем считать, что пользовательский тип успешно определен и все диалоги могут быть закрыты.

После создания пользовательского типа данных можно возвратиться на уровень процесса и определить оставшийся элемент данных «Order Status». В уже известном диалоговом окне «Workflow relevant data — defining» вам потребуется выполнить те же действия по определению элементов данных, что и раньше, за исключением того, что теперь нужно выбрать тип данных «Declared», а подтип — «Order Status Type» (рис. 2.25).

Теперь все требуемые элементы данных процесса определены, и мы можем уточнить, с помощью какого приложения в функции «получение заказа» эти элементы данных получают свое значение.

#### *2.4.7. Шаг 6. Работа над бизнес-процессом: определение приложения для исполнения функции «получение заказа»*

Для выполнения функции «получение заказа» определяем одно инструментальное средство с условным названием «GetOrderDetails». Двойным нажатием левой клавиши мыши вызываем диалог для определения параметров функции и переходим на закладку «Tools». Для добавления

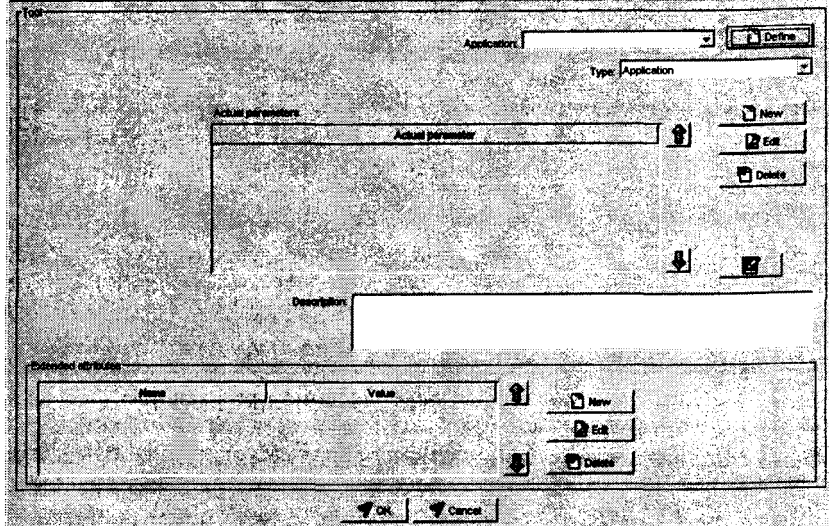


Рис. 2.26. Диалог для перечисления инструментальных средств исполнения функции

нового инструментального средства нажимаем кнопку «New» — появляется диалог «Tool defining» (рис. 2.26).

В этом диалоге указываем, что инструментальное средство является приложением (элемент «Application» из списка «Type»). Так как ни одного приложения у нас не определено, потребуется его заново определить. Для определения нового приложения нужно нажать кнопку «Define». Появится диалог «Applications defining». В нем нажимаем кнопку «New» (рис. 2.27).

В результате нажатия кнопки «New» возникнет заключительный диалог в этой цепочке — «Application defining» (рис. 2.28).

В этом диалоге нужно определить имя нового приложения в поле «Name» (в данном случае — это «GetOrderDetails») и перечень формальных параметров этого приложения. Формальными параметрами приложения «GetOrderDetails» должны являться:



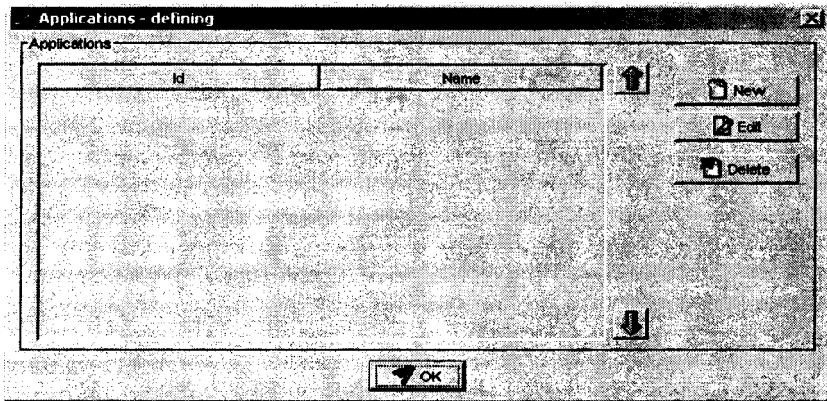


Рис. 2.27. Диалог для перечисления приложений

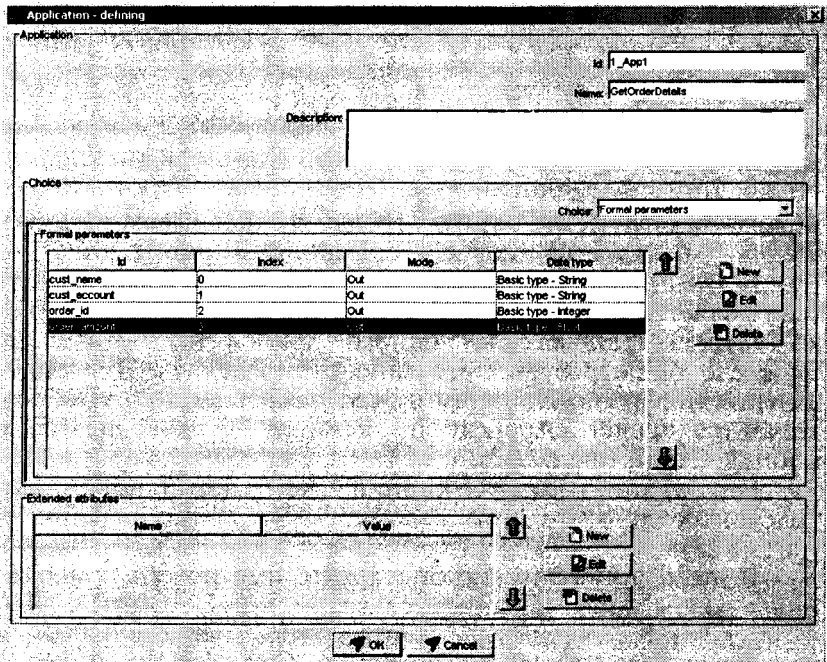
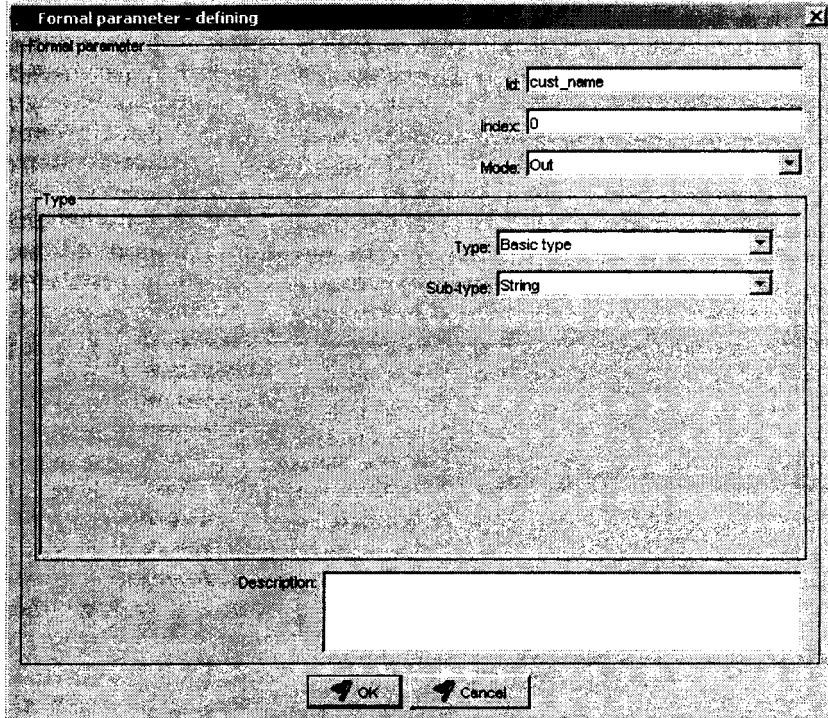


Рис. 2.28. Диалог для определения нового приложения





**Рис. 2.29.** Диалог для определения нового формального параметра приложения


1. `cust_name` — выходной параметр типа `String`;
2. `cust_account` — выходной параметр типа `String`;
3. `order_id` — выходной параметр типа `Integer`;
4. `order_amount` — выходной параметр типа `Float`.

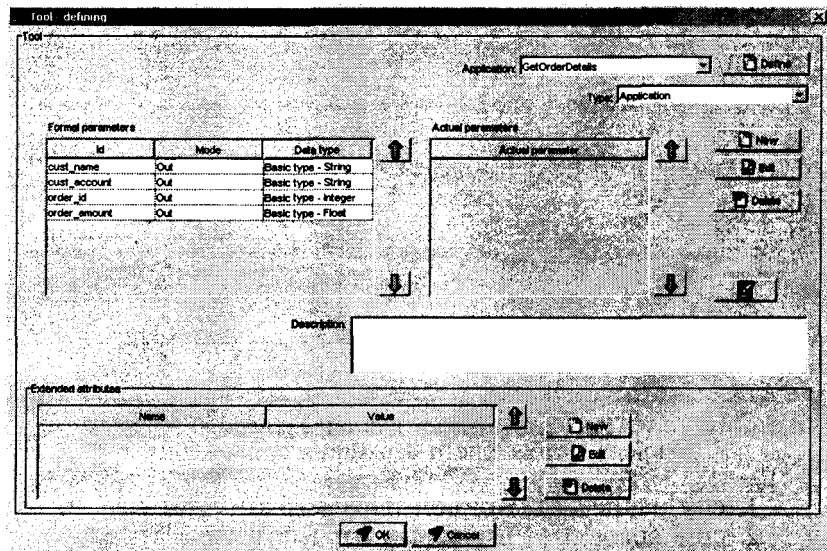
Формальные параметры приложения добавляются последовательно путем нажатия кнопки «New». Каждый раз по нажатию кнопки «New» появляется диалог «Formal parameter — defining» (рис. 2.29).

В этом диалоге указываются идентификатор параметра, его порядковый номер, роль (входной — IN, или выходной — OUT) и тип данных. После определения перечня формальных параметров приложения его объявление можно считать законченным. Необходимо закрыть диалог и

возвратиться на уровень предыдущего диалогового окна «Tool — defining» для того, чтобы связать формальные параметры приложения с фактическими параметрами вызова в контексте данной функции<sup>9</sup>.

В диалоговом окне «Tool — defining» для этого нужно нажать на кнопку «View Formal parameters for selected item»

. В результате слева, в области «Formal parameters», появится список формальных параметров текущего приложения (рис. 2.30).



**Рис. 2.30.** Диалог для перечисления инструментальных средств исполнения функции с указанием формальных параметров

В правой области «Actual parameters» нужно построить соответствующий список, состоящий из ранее определенных элементов данных процесса. При вызове приложения в

<sup>9</sup> Фактические параметры — это определенные элементы данных бизнес-процесса (Workflow Relevant Data). При выполнении одного и того же приложения в разных функциях с формальными параметрами приложения могут быть связаны разные элементы данных.

контексте текущей функции вместо каждого формального параметра из области «Formal parameters» будет подставляться значение элемента данных процесса, находящегося на той же строке в области «Actual parameters». Поэтому мы должны определить такое соответствие формальных и фактических параметров:

- ◆ формальный параметр «cust\_name» соответствует элементу данных «Customer\_Name»;
- ◆ формальный параметр «cust\_account» соответствует элементу данных «Account\_Number»;
- ◆ формальный параметр «order\_id» соответствует элементу данных «Order\_Number»;
- ◆ формальный параметр «order\_amount» соответствует элементу данных «Order\_Amount».

Добавление элементов данных в область «Actual parameters» производится кнопкой «New».

После того, как список фактических параметров вызова приложения сформирован, можно закрыть все диалоговые окна и возвратиться в основное окно редактирования бизнес-процесса.

#### *2.4.8. Шаг 7. Работа над бизнес-процессом: определение подпроцесса «Проверка платежеспособности» (Check\_Credit)*

Для создания этой функции используется инструмент «SubFlow» (подпроцесс) из панели инструментов «Toolbox». Выберем этот инструмент на панели и перенесем пиктограмму подпроцесса на «swim-line» первого участника правее функции «ReceiveOrder».

Для типа функции «Subflow» редактирование внутреннего содержимого или параметров возможно только через команду «Edit» (или «Properties») контекстного меню. Поэтому перемещаем курсор мыши на пиктограмму функции «Subflow» и нажимаем правую кнопку мыши. Появляется контекстное меню, из которого выбираем команду «Edit». При этом появляется новое окно для редактирования подпроцесса (рис. 2.31).

В этом подпроцессе, прежде всего, создаем участника типа «подразделение организации» (Organizational unit) с идентификатором «Organization». Далее определяем в под-

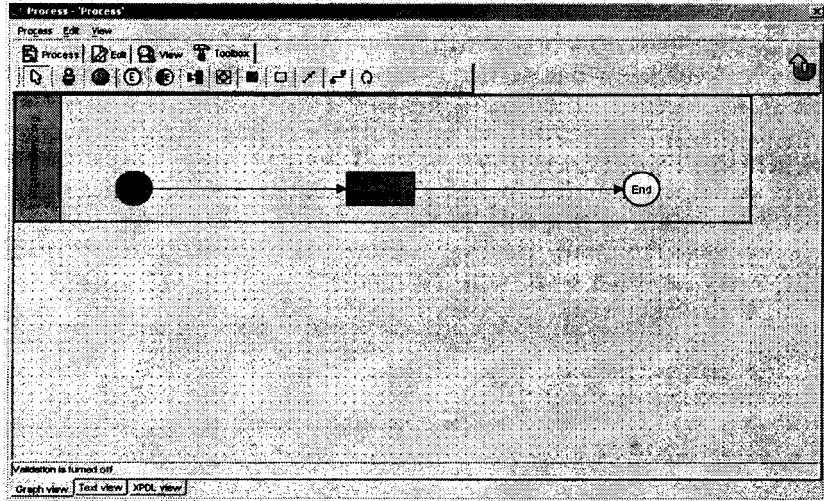


Рис. 2,31. Окно для определения подпроцесса

процессе старт-стоп процесса и функцию «Проверка» («Check») <sup>10</sup>.

Для продолжения работы нам потребуется определить формальные параметры этого подпроцесса. Эти параметры будут инициализироваться какими-то конкретными значениями при вызове из процесса уровнем выше. Таким образом, наш подпроцесс будет являться аналогом подпрограммы с параметрами. Делаем активной панель инструментов «Process» и выбираем на ней инструмент «Formal parameters». При этом появляется диалоговое окно «Process 'Process' — Formal Parameters» (рис. 2.32).

Подобно тому, как мы определяли ранее формальные параметры приложения, в данном случае нам потребуется определить следующие формальные параметры подпроцесса:

1. Customer\_Account — входной параметр типа String;
2. Order\_Amount — входной параметр типа Float;

<sup>10</sup> Действия по переопределению параметров функции Check аналогичны рассмотренным ранее и здесь не рассматриваются.

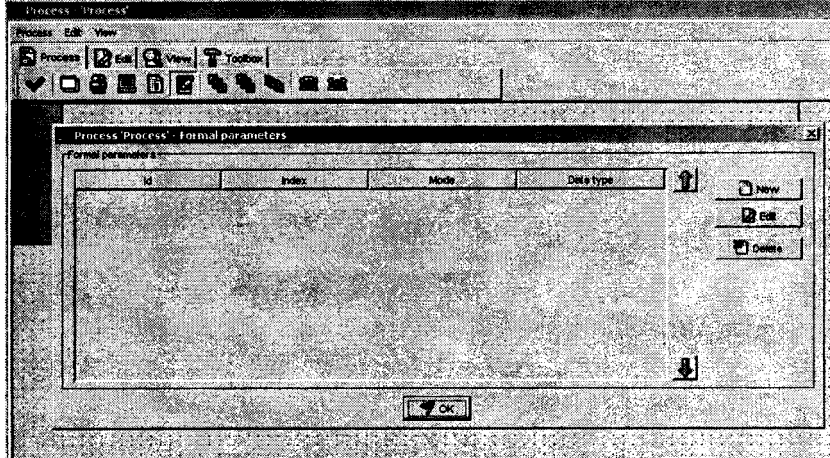


Рис. 2.32. Диалог перечисления формальных параметров подпроцесса

3. `Order_Status` — выходной параметр перечислимого типа `{OK, NOT_OK}`.

Определение формального параметра производится нажатием кнопки «New». Появляется уже известное окно для определения формального параметра. Мы приведем здесь пример лишь диалога для определения третьего формального параметра (рис. 2.33).

Для полного определения подпроцесса остается создать приложение (пусть оно будет иметь название «DoCheck»), которое будет исполнять функцию «Check», и настроить параметры вызова приложения в свойствах функции «Check». Приложение «DoCheck» должно иметь три параметра:

- 1) `account` — входной параметр типа `String`;
- 2) `amount` — входной параметр типа `Float`;
- 3) `status` — выходной параметр перечислимого типа `{OK, NOT_OK}`.

Действия по определению приложения аналогичны описанным на шаге 6, поэтому здесь мы не приводим детали. Обратите внимание на правильность сопоставления формальных параметров приложения фактическим параметрам вызова в функции «Check». В данном случае фор-

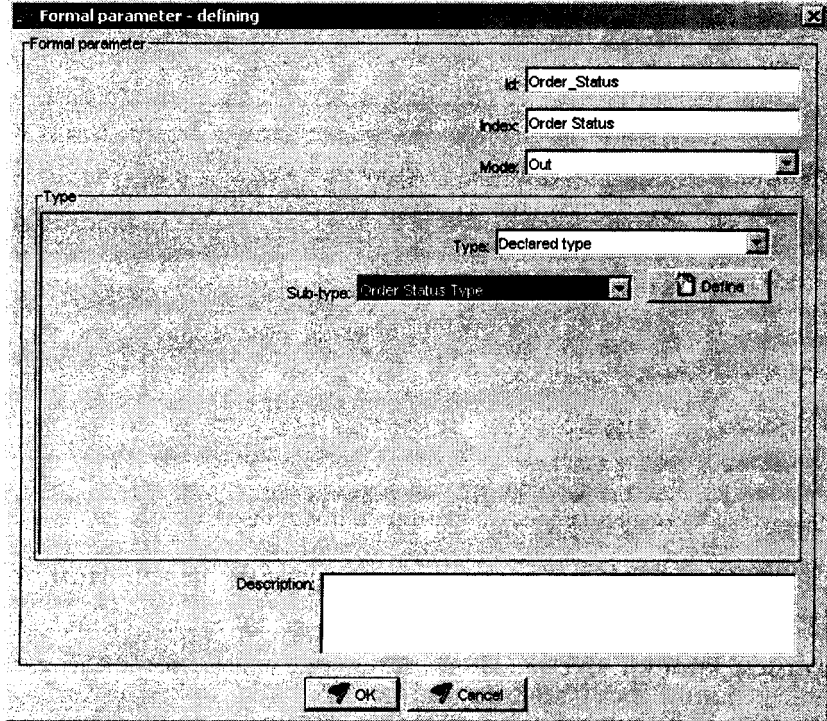


Рис. 2.33. Диалог для определения нового формального параметра подпроцесса

мальные параметры приложения «DoCheck» должны сопоставляться не с элементами данных подпроцесса (у данного подпроцесса их нет), а с формальными параметрами подпроцесса (рис. 2.34).

В заключение определения подпроцесса стоит изменить его название по умолчанию, вызвав инструмент «Process properties» из панели инструментов «Process». В появившемся диалоговом окне потребуется изменить поля «Id» и «Name» (рис. 2.35).

На этом определение подпроцесса можно считать законченным, и окно определения процесса может быть закрыто.

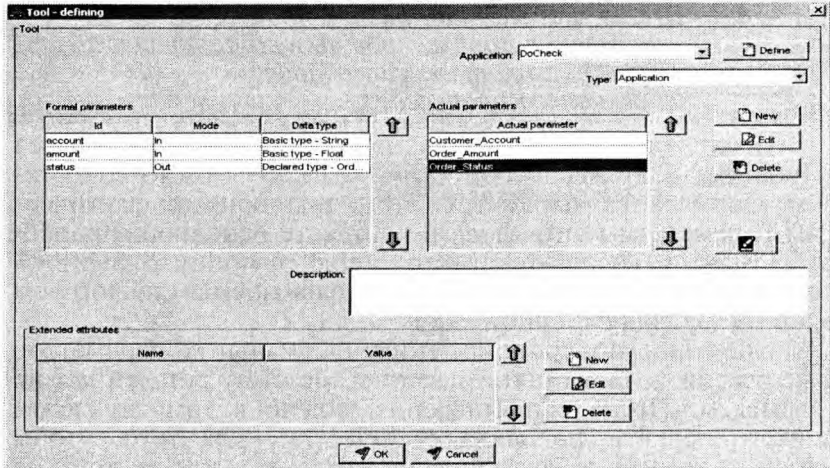


Рис. 2.34. Диалог для перечисления инструментальных средств исполнения функции с указанием формальных и фактических параметров

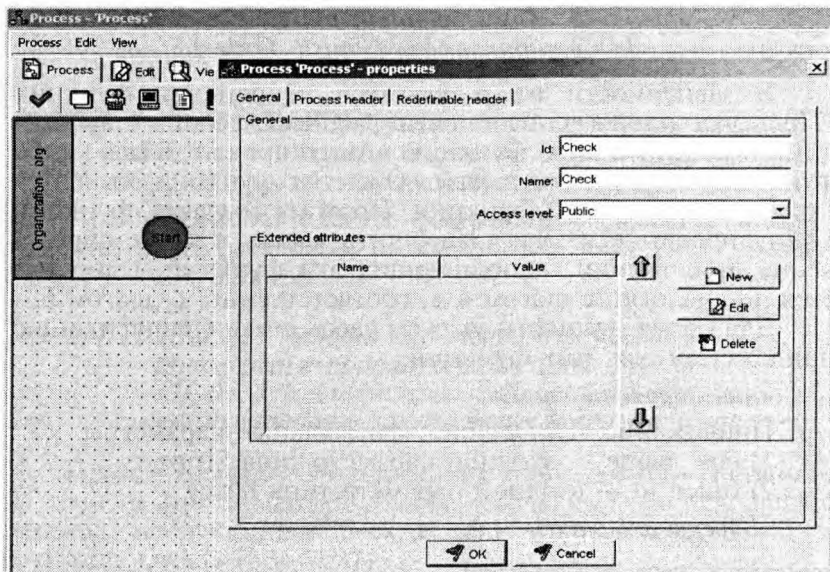


Рис. 2.35. Изменение имени и идентификатора подпроцесса

#### **2.4.9. Шаг 8. Работа над бизнес-процессом: определение параметров вызова подпроцесса «Проверка платежеспособности» из основного процесса**

После определения всех подробностей подпроцесса «Проверка платежеспособности» («Check») можно сопоставить формальные параметры этого подпроцесса фактическим параметрам при вызове в контексте основного процесса<sup>11</sup>. Вызвав из контекстного меню функции «Subflow» команду «Properties», перейдем в появившемся диалоговом окне на закладку «Subflow» (рис. 2.36).

Операции по сопоставлению формальных параметров фактически должны быть известны, поэтому деталей мы не указываем. После сопоставления можно в том же самом диалоге перейти на закладку «General» для того, чтобы изменить название функции (а также идентификатор) на «CheckAccount».

#### **2.4.10. Шаг 9. Работа над бизнес-процессом: создание функций «Аннулировать заказ» (Decline) и «Завершить заказ» (Finish\_Order)**

В зависимости от результатов исполнения функции «Проверка платежеспособности» («CheckAccount») необходимо выполнить либо функцию «Аннулировать заказ», либо функцию «Завершить заказ». Обе эти функции являются экземплярами общей функции. Поэтому создание функций «Аннулировать заказ» и «Завершить заказ», а также определение приложений, исполняющих эти функции, выполняется по аналогии с шагом 4 и, соответственно, с шагом 6.

Для определенности укажем здесь лишь спецификацию приложений для этих функций.

##### **«Аннулировать заказ»**

Приложение «DoDecline», формальные параметры:

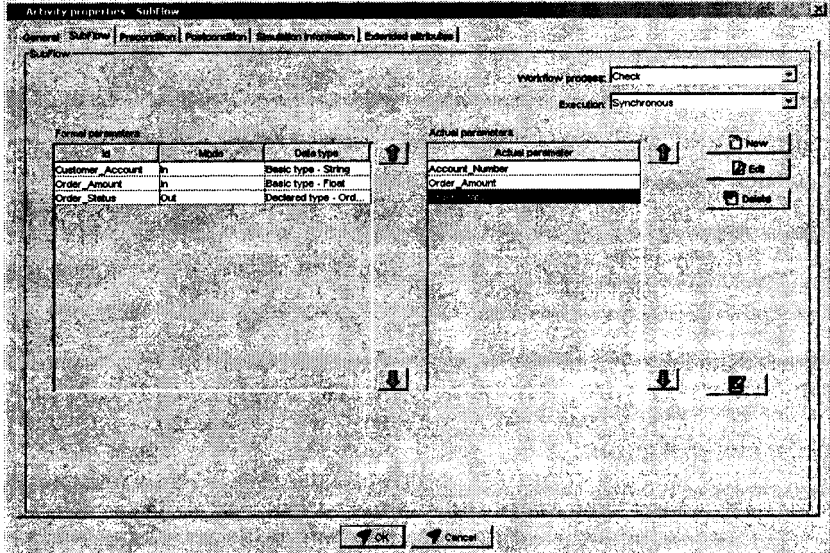
- 1) cust\_name — входной параметр типа String;
- 2) order\_id — входной параметр типа String.

##### **«Завершить заказ»**

---

<sup>11</sup> Фактическими параметрами, в данном случае, должны являться элементы данных процесса.





**Рис. 2.36.** Диалог для сопоставления формальных параметров подпроцесса с фактическими параметрами при вызове из текущей функции

Приложение «DoFinish», формальные параметры:

- 1) cust\_name — входной параметр типа String;
- 2) order\_id — входной параметр типа String.

Правила сопоставления формальных параметров приложений с фактическими параметрами, пожалуйста, разработайте самостоятельно.

#### *2.4.11. Шаг 10. Работа над бизнес-процессом: определение правил перехода от функции «Проверка платежеспособности» к функции «Аннулировать заказ» или «Завершить заказ»*

Прежде всего, с использованием инструмента «Transition» из панели инструментов «Tools» создадим два исходящих перехода из функции «CheckAccount» к функциям «Decline» и «Finish\_Order» (рис. 2.37).

Теперь надо определить ограничения на эти исходящие переходы (в терминах редактора JaWE —

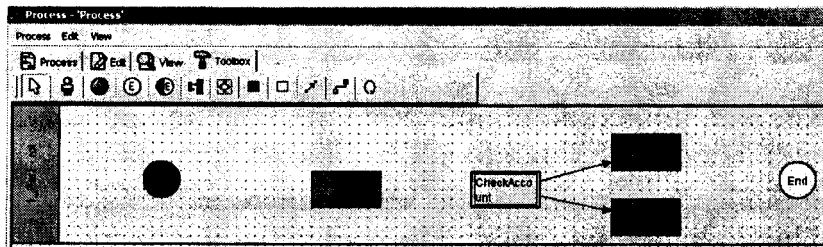


Рис. 2.37. Связь переходами функции «CheckAccount» с функциями «Decline» и «Finish\_Order»

«Postcondition») для функции «CheckAccount». Это нужно сделать из-за того, что в результате выполнения подпроцесса «Check» возможно два взаимоисключающих варианта маршрутизации:

1) кредитоспособность подтверждается — формальный параметр Order\_Status имеет значение OK;

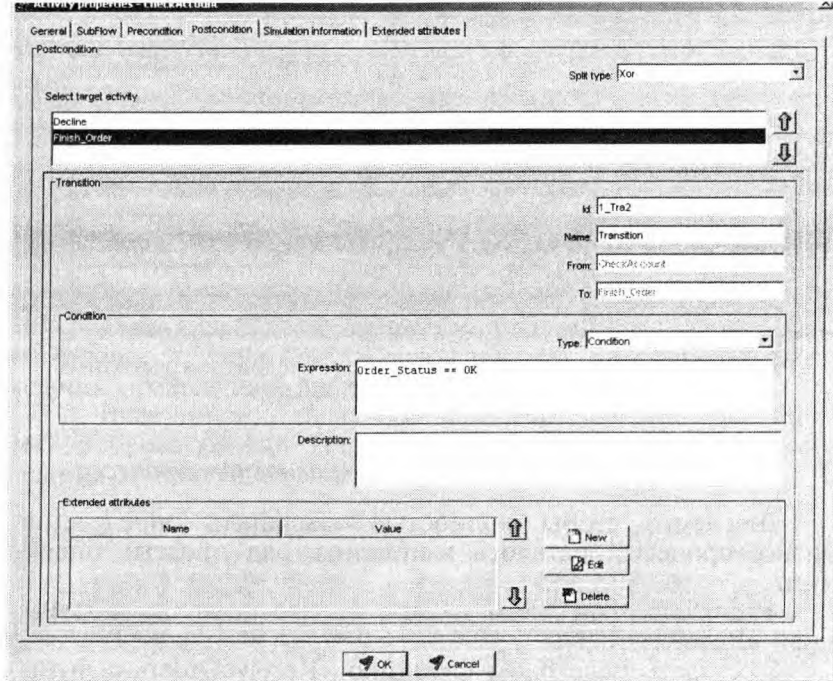
2) кредитоспособность не подтверждается — формальный параметр Order\_Status имеет значение NOT\_OK.

В первом случае необходимо исполнить функцию по обработке заказа («Finish\_Order»), а во втором случае — аннулировать заказ («Decline»).

Для определения ограничений на исходящие переходы вызываем из контекстного меню функции «CheckAccount» команду Properties и в появившемся диалоговом окне переходим на закладку «Postcondition» (рис. 2.38).

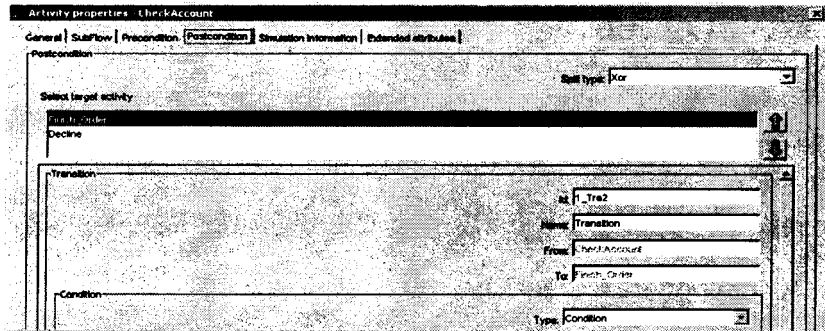
Сначала определим, что переход к функции «Finish\_Order» должен происходить лишь тогда, когда элемент данных процесса Order\_Status имеет значение OK. Для этого в списке «Select Target Activities» выделяем нужную функцию — адресат, а в поле «Expression» записываем требуемое условие. Поскольку при определении пакета мы указали, что в записи выражений будет использоваться синтаксис языка Javascript, введенное нами выражение должно удовлетворять синтаксическим правилам языка Javascript.

В заключение переходу на функцию «Decline» установим тип «Otherwise», выбрав это значение из списка «Type» (рис. 2.39).



**Рис. 2.38.** Диалог для определения правил срабатывания переходов от функции «CheckAccount» к функции «Decline» или «Finish\_Order»

Обратим внимание еще на одно изменение в исходном описании бизнес-процесса. Во время работы с определением процесса выяснилось, что нужно строго следить за правильным порядком следования исходящих переходов в определении функций. Например, в первоначальном варианте для функции «check credit» в начале идет выходной переход с типом «OTHERWISE». Если оставить такой порядок без изменения, то всегда будет срабатывать именно этот переход и, соответственно, исполняться функция «decline». Поэтому порядок нужно поменять так, чтобы в начале списка переходов стоял переход на функцию «Finish\_Order» с «CONDITION» (рис. 2.39).



**Рис. 2.39.** JaWE — правильный порядок следования выходных переходов в функции «Check»

#### *2.4.12. Шаг 11. Работа над бизнес-процессом: уточнение всех деталей в основном процессе*

Для того чтобы полностью завершить определение бизнес-процесса, осталось выполнить ряд простых операций:

- 1) связать переходом начало бизнес-процесса с функцией «ReceiveOrder»;
- 2) связать переходом функцию «ReceiveOrder» с функцией «CheckAccount»;
- 3) добавить функцию типа «маршрутизатор» и связать ее входными переходами с функциями «Decline» и «Finish\_Order»;
- 4) связать переходом функцию-маршрутизатор с окончанием бизнес-процесса.

При этом никаких дополнительных действий по изменению свойств функций не требуется. Процесс в итоге должен выглядеть так, как показано на рис. 2.10, 2.12.

В заключение можно проверить логическую правильность созданного описания, вызвав из основного меню команду «Process/Check Validity». Если в ходе проверки ошибок не обнаружено, то можно закрыть окно определения процесса и перейти на уровень пакета.

На уровне пакета остается записать все сделанные изменения в XPDL-файл, вызвав команду «File/Save». Ре-

датор выполнит полную проверку определения пакета и сохранит изменения.

Нужно отметить, что проделанную работу в редакторе JaWE можно представить в виде процесса, изображенного на рис. 2.40.

## 2.5. Контрольные задания к главе 2

1. Доведите в редакторе JaWE определение бизнес-процесса «обработка заказа» до первоначального варианта, представленного в предыдущей главе.

2. Выполните в редакторе JaWE полное описание бизнес-процесса уровня 1 вашего варианта из практического задания предыдущей главы.

3. Выполните в редакторе JaWE полное описание бизнес-процесса уровня 2 вашего варианта из практического задания предыдущей главы.

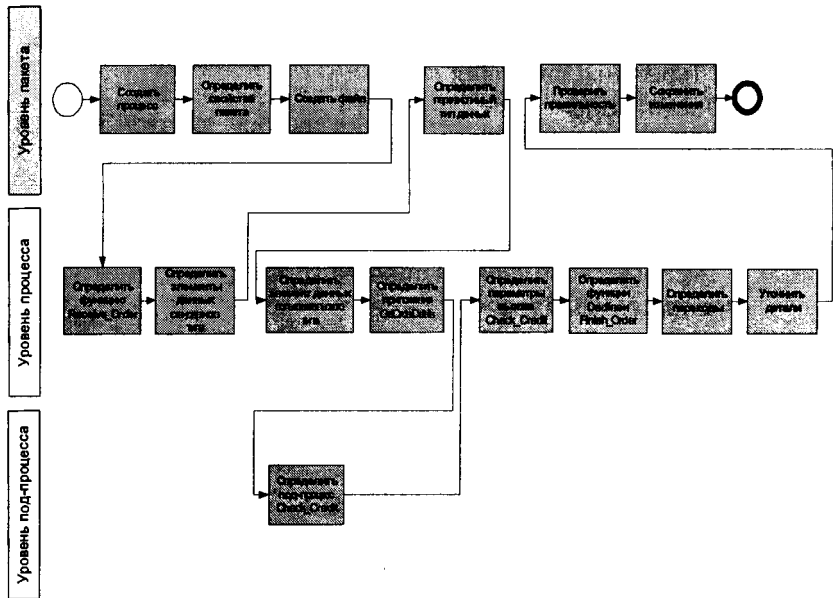


Рис. 2.40. «Процесс» определения процесса в редакторе JaWE

## 2.6. Приложение к главе 2

Содержимое XPDL-файла для примера, разобранный в главе 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<Package Id="order" xmlns="http://www.wfmc.org/2002/XPDL1.0"
xmlns:xpdl="http://www.wfmc.org/2002/XPDL1.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.wfmc.org/2002/XPDL1.0 http://
wfmc.org/standards/docs/TC-1025_schema_10_xpdl.xsd">
  <PackageHeader>
    <XPDLVersion>1.0</XPDLVersion>
    <Vendor>Together</Vendor>
    <Created>2005-01-04 10:24:48</Created>
  </PackageHeader>
  <RedefinableHeader PublicationStatus="UNDER_TEST"/>
  <ConformanceClass GraphConformance="NON_BLOCKED"/>
  <Script Type="text/javascript"/>
  <TypeDeclarations>
    <TypeDeclaration Id="Order_Status_Type" Name="Order Status
Type">
      <EnumerationType>
        <EnumerationValue Name="OK"/>
        <EnumerationValue Name="NOT_OK"/>
      </EnumerationType>
    </TypeDeclaration>
  </TypeDeclarations>
  <WorkflowProcesses>
    <WorkflowProcess AccessLevel="PUBLIC" Id="1"
Name="Process">
      <ProcessHeader DurationUnit="D">
        <Created>2005-01-04 10:30:08</Created>
      </ProcessHeader>
      <RedefinableHeader PublicationStatus="UNDER_TEST"/>
      <DataFields>
```

```
<DataField Id="Order_Number" IsArray="FALSE"
Name="Order Number">
  <DataType>
    <BasicType Type="INTEGER"/>
  </DataType>
</DataField>
<DataField Id="Customer_Name" IsArray="FALSE"
Name="Customer Name">
  <DataType>
    <BasicType Type="STRING"/>
  </DataType>
</DataField>
<DataField Id="Order_Amount" IsArray="FALSE"
Name="Order Amount">
  <DataType>
    <BasicType Type="FLOAT"/>
  </DataType>
</DataField>
<DataField Id="Account_Number" IsArray="FALSE"
Name="Account Number">
  <DataType>
    <BasicType Type="STRING"/>
  </DataType>
</DataField>
<DataField Id="Order_Status" IsArray="FALSE"
Name="Order Status">
  <DataType>
    <DeclaredType Id="Order_Status_Type"/>
  </DataType>
</DataField>
</DataFields>
<Participants>
  <Participant Id="1_Par1">
    <ParticipantType Type="ROLE"/>
  </Participant>
</Participants>
```

<Applications>

<Application Id="1\_App1" Name="GetOrderDetails">

<FormalParameters>

<FormalParameter Id="cust\_name" Index="0"

Mode="OUT">

<DataType>

<BasicType Type="STRING"/>

</DataType>

</FormalParameter>

<FormalParameter Id="cust\_account" Index="1"

Mode="OUT">

<DataType>

<BasicType Type="STRING"/>

</DataType>

</FormalParameter>

<FormalParameter Id="order\_id" Index="2"

Mode="OUT">

<DataType>

<BasicType Type="INTEGER"/>

</DataType>

</FormalParameter>

<FormalParameter Id="order\_amount" Index="3"

Mode="OUT">

<DataType>

<BasicType Type="FLOAT"/>

</DataType>

</FormalParameter>

</FormalParameters>

</Application>

<Application Id="DoDecline" Name="DoDecline">

<FormalParameters>

<FormalParameter Id="cust\_name" Mode="IN">

<DataType>

<BasicType Type="STRING"/>

</DataType>



```
</FormalParameter>
  <FormalParameter Id="order_id" Index="order_id"
Mode="IN">
    <DataType>
      <BasicType Type="STRING"/>
    </DataType>
  </FormalParameter>
</FormalParameters>
</Application>
<Application Id="DoFinish">
  <FormalParameters>
    <FormalParameter Id="cust_name" Mode="IN">
      <DataType>
        <BasicType Type="STRING"/>
      </DataType>
    </FormalParameter>
    <FormalParameter Id="order_id" Mode="IN">
      <DataType>
        <BasicType Type="STRING"/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
</Application>
</Applications>
<Activities>
  <Activity Id="1_Act1" Name="ReceiveOrder">
    <Implementation>
      <Tool Id="1_App1" Type="APPLICATION">
        <ActualParameters>
          <ActualParameter>Customer_Name
</ActualParameter>
          <ActualParameter>Account_Number
</ActualParameter>
          <ActualParameter>Order_Number
</ActualParameter>
```

```
                <ActualParameter>Order_Amount
</ActualParameter>
            </ActualParameters>
        </Tool>
    </Implementation>
    <Performer>1_Par1</Performer>
    <StartMode>
        <Automatic/>
    </StartMode>
    <FinishMode>
        <Automatic/>
    </FinishMode>
    <ExtendedAttributes>
        <ExtendedAttribute Name="ParticipantID"
Value="1_Par1"/>
        <ExtendedAttribute Name="XOffset" Value="210"/>
        <ExtendedAttribute Name="YOffset" Value="70"/>
    </ExtendedAttributes>
</Activity>
<Activity Id="CheckAccount" Name="CheckAccount">
    <Implementation>
        <SubFlow Execution="SYNCHR" Id="Check">
            <ActualParameters>
                <ActualParameter>Account_Number
</ActualParameter>
                <ActualParameter>Order_Amount
</ActualParameter>
                <ActualParameter>Order_Status
</ActualParameter>
            </ActualParameters>
        </SubFlow>
    </Implementation>
    <StartMode>
        <Automatic/>
    </StartMode>
```

<FinishMode>

<Automatic/>

</FinishMode>

<TransitionRestrictions>

<TransitionRestriction>

<Split Type="XOR">

<TransitionRefs>

<TransitionRef Id="1\_Tra1"/>

<TransitionRef Id="1\_Tra2"/>

</TransitionRefs>

</Split>

</TransitionRestriction>

</TransitionRestrictions>

<ExtendedAttributes>

<ExtendedAttribute Name="ParticipantID"

Value="1\_Par1"/>

<ExtendedAttribute Name="XOffset" Value="380"/>

<ExtendedAttribute Name="YOffset" Value="70"/>

</ExtendedAttributes>

</Activity>

<Activity Id="Decline" Name="Decline">

<Implementation>

<Tool Id="DoDecline" Type="APPLICATION">

<ActualParameters>

<ActualParameter>Customer\_Name

</ActualParameter>

<ActualParameter>Order\_Number

</ActualParameter>

</ActualParameters>

</Tool>

</Implementation>

<Performer>1\_Par1</Performer>

<StartMode>

<Automatic/>

</StartMode>

```
<FinishMode>
  <Automatic/>
</FinishMode>
<ExtendedAttributes>
  <ExtendedAttribute Name="ParticipantID"
Value="1_Par1"/>
  <ExtendedAttribute Name="XOffset" Value="530"/>
  <ExtendedAttribute Name="YOffset" Value="30"/>
</ExtendedAttributes>
</Activity>
<Activity Id="Finish_Order" Name="Finish_Order">
  <Implementation>
    <Tool Id="DoFinish" Type="APPLICATION">
      <ActualParameters>
        <ActualParameter>Customer_Name
</ActualParameter>
        <ActualParameter>Order_Number
</ActualParameter>
      </ActualParameters>
    </Tool>
  </Implementation>
  <Performer>1_Par1</Performer>
  <StartMode>
    <Automatic/>
  </StartMode>
  <FinishMode>
    <Automatic/>
  </FinishMode>
  <ExtendedAttributes>
    <ExtendedAttribute Name="ParticipantID"
Value="1_Par1"/>
    <ExtendedAttribute Name="XOffset" Value="530"/>
    <ExtendedAttribute Name="YOffset" Value="100"/>
  </ExtendedAttributes>
</Activity>
```

<Activity Id="1\_Act5" Name="Route">

<Route/>

<StartMode>

<Automatic/>

</StartMode>

<FinishMode>

<Automatic/>

</FinishMode>

<TransitionRestrictions>

<TransitionRestriction>

<Join Type="XOR"/>

</TransitionRestriction>

</TransitionRestrictions>

<ExtendedAttributes>

<ExtendedAttribute Name="ParticipantID"

Value="1\_Par1"/>

<ExtendedAttribute Name="XOffset" Value="670"/>

<ExtendedAttribute Name="YOffset" Value="60"/>

</ExtendedAttributes>

</Activity>

</Activities>

<Transitions>

<Transition From="CheckAccount" Id="1\_Tra1"

Name="Transition" To="Decline">

<Condition Type="OTHERWISE"/>

<ExtendedAttributes>

<ExtendedAttribute Name="RoutingType"

Value="NOROUTING"/>

</ExtendedAttributes>

</Transition>

<Transition From="CheckAccount" Id="1\_Tra2"

Name="Transition" To="Finish\_Order">

<Condition Type="CONDITION">Order\_Status ==

OK</Condition>

<ExtendedAttributes>

```
        <ExtendedAttribute Name="RoutingType"
Value="NOROUTING"/>
    </ExtendedAttributes>
</Transition>
    <Transition From="1_Act1" Id="1_Tra4" Name="Transition"
To="CheckAccount">
        <ExtendedAttributes>
            <ExtendedAttribute Name="RoutingType"
Value="NOROUTING"/>
        </ExtendedAttributes>
    </Transition>
    <Transition From="Decline" Id="1_Tra6"
Name="Transition" To="1_Act5">
        <ExtendedAttributes>
            <ExtendedAttribute Name="RoutingType"
Value="NOROUTING"/>
        </ExtendedAttributes>
    </Transition>
    <Transition From="Finish_Order" Id="1_Tra7"
Name="Transition" To="1_Act5">
        <ExtendedAttributes>
            <ExtendedAttribute Name="RoutingType"
Value="NOROUTING"/>
        </ExtendedAttributes>
    </Transition>
</Transitions>
    <ExtendedAttributes>
        <ExtendedAttribute Name="StartOfWorkflow"
Value="1_Par1;1_Act1;130;70;NOROUTING"/>
        <ExtendedAttribute Name="EndOfWorkflow"
Value="1_Par1;1_Act5;820;60;NOROUTING"/>
        <ExtendedAttribute Name="ParticipantVisualOrder"
Value="1_Par1;"/>
    </ExtendedAttributes>
</WorkflowProcess>
```

```
<WorkflowProcess AccessLevel="PUBLIC" Id="Check"
Name="Check">
  <ProcessHeader DurationUnit="D">
    <Created>2005-01-04 11:53:32</Created>
  </ProcessHeader>
  <RedefinableHeader PublicationStatus="UNDER_TEST"/>
  <FormalParameters>
    <FormalParameter Id="Customer_Account"
Index="Customer_Account" Mode="IN">
      <DataType>
        <BasicType Type="STRING"/>
      </DataType>
    </FormalParameter>
    <FormalParameter Id="Order_Amount" Index="Order
Amount" Mode="IN">
      <DataType>
        <BasicType Type="FLOAT"/>
      </DataType>
    </FormalParameter>
    <FormalParameter Id="Order_Status" Index="Order
Status" Mode="OUT">
      <DataType>
        <DeclaredType Id="Order_Status_Type"/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
  <Participants>
    <Participant Id="Organization">
      <ParticipantType Type="ORGANIZATIONAL_UNIT"/>
    </Participant>
  </Participants>
  <Applications>
    <Application Id="DoCheck" Name="DoCheck">
      <FormalParameters>
        <FormalParameter Id="account" Mode="IN">
```

```
        <DataType>
            <BasicType Type="STRING"/>
        </DataType>
    </FormalParameter>
    <FormalParameter Id="amount" Mode="IN">
        <DataType>
            <BasicType Type="FLOAT"/>
        </DataType>
    </FormalParameter>
    <FormalParameter Id="status" Mode="OUT">
        <DataType>
            <DeclaredType Id="Order_Status_Type"/>
        </DataType>
    </FormalParameter>
</FormalParameters>
</Application>
</Applications>
<Activities>
    <Activity Id="Check" Name="Check">
        <Implementation>
            <Tool Id="DoCheck" Type="APPLICATION">
                <ActualParameters>
                    <ActualParameter>Customer_Account
</ActualParameter>
                    <ActualParameter>Order_Amount
</ActualParameter>
                    <ActualParameter>Order_Status
</ActualParameter>
                </ActualParameters>
            </Tool>
        </Implementation>
        <Performer>Organization</Performer>
        <StartMode>
            <Automatic/>
        </StartMode>
```



```
<FinishMode>
  <Automatic/>
</FinishMode>
<ExtendedAttributes>
  <ExtendedAttribute Name="ParticipantID"
Value="Organization"/>
  <ExtendedAttribute Name="XOffset" Value="360"/>
  <ExtendedAttribute Name="YOffset" Value="90"/>
</ExtendedAttributes>
```

```
</Activity>
```

```
</Activities>
```

```
<ExtendedAttributes>
```

```
<ExtendedAttribute Name="StartOfWorkflow"
Value="Organization;Check;110;90;NOROUTING"/>
```

```
<ExtendedAttribute Name="EndOfWorkflow"
```

```
Value="Organization;Check;660;90;NOROUTING"/>
```

```
<ExtendedAttribute Name="ParticipantVisualOrder"
```

```
Value="Organization;"/>
```

```
</ExtendedAttributes>
```

```
</WorkflowProcess>
```

```
</WorkflowProcesses>
```

```
<ExtendedAttributes>
```

```
<ExtendedAttribute Name="MadeBy" Value="JaWE"/>
```

```
<ExtendedAttribute Name="Version" Value="1.2"/>
```

```
</ExtendedAttributes>
```

```
</Package>
```

---

## Глава 3 \_\_\_\_\_

### Распорядитель выполнения работ

В предыдущей главе мы познакомились с редактором спецификаций процессов JaWE. Он позволяет создавать определение бизнес-процесса в графическом виде, экспортировать и импортировать определения на языке XPDЛ.

Созданные определения бизнес-процесса, включающие в себя отдельные функции, переходы, описание участников, приложения и данные, могут быть загружены в особую программную систему — систему workflow management. Она позволяет производить настройку абстрактных участников и приложений, связывая их с конкретными сущностями, создавать экземпляры процессов. Такая система обязательно содержит особый компонент — распорядитель выполнения работ (workflow Engine). Под его контролем создаются и запускаются на выполнение экземпляры процессов, вызываются внешние приложения для исполнения функций, идет управление ходом процесса.

Для знакомства с возможностями и архитектурой таких систем, а также средствами их администрирования рассмотрим программную систему Shark.

#### *3.1. Общая информация о системе Shark*

Shark является набором библиотек и приложений на языке Java, разрабатываемых и свободно распространяемых консорциумом ObjectWeb (домашняя страница в Интернете — <http://shark.objectweb.org>). Shark относится к так называемым программным продуктам «Back End». Это означает, что основная часть его функциональности относится не к

интерфейсу пользователя, а к внутренним алгоритмам, которые обеспечивают выполнение и управление бизнес-процессами, описанными на языке XPDЛ и предварительно загруженными в Shark. Отдельные графические приложения, входящие в состав Shark, можно рассматривать как «верхушку айсберга», они являются вспомогательным средством администрирования. Сам распорядитель выполнения работ (workflow engine) может работать без них.

### *Отличительные особенности Shark*

1. Shark построен в соответствии с рекомендациями и стандартами консорциума WfMC. Язык XPDЛ является исходным форматом для представления бизнес-процессов внутри распорядителя выполнением работ, а в системе реализованы все стандартные интерфейсы. Причем они опубликованы, поэтому различные внешние приложения других разработчиков могут обращаться за информацией по статистике выполнения процессов к системе Shark.

1. Набор графических приложений, входящих в состав Shark, позволяет выполнять все администраторские обязанности (управление процессами, настройка, контроль и т.п.). Причем эти приложения могут работать в удаленном режиме, когда распорядитель работает на одном компьютере, а пользователь приложения — на другом.

2. Большая часть алгоритмов и структур данных реализована в виде библиотеки на языке Java. Поэтому Shark может работать на различных платформах (операционная система, процессор), а также быть встроенным в состав другой системы. Так, Shark встроен в ERP-систему Ofbiz, с которой мы познакомимся позднее. В этой системе Shark обеспечивает возможность управления бизнес-процессами в соответствии со стандартами WfMC.

3. В реализации системы Shark активно использованы различные средства динамического конфигурирования, расширения и переопределения большинства базовых алгоритмов и структур данных. Значения параметров легко изменяются в файлах конфигурации или через администраторские приложения. Вызов многих методов классов в библиотеке происходит косвенно через соответствующие методы интер-

фейсов. А так как язык Java позволяет динамически по символическому имени загружать различные классы, реализующие один и тот же интерфейс, то появляется возможность полностью или частично изменить поведение любого метода стандартного интерфейса. Прежде всего, это необходимо для подключения к распорядителю работ различных, заранее неизвестных алгоритмов и внешних программных систем, играющих роль приложений в бизнес-процессе. Используя возможности по расширению функциональности, администратор может легко обеспечить доступ из функций бизнес-процесса к любым источникам данных и информационным системам предприятия (электронная почта, алгоритмы на языке Java, удаленные системы, стандартные программы и т.п.).

4. Алгоритмы и структуры данных в распорядителе работ устойчивы к аппаратным и программным сбоям: все описания бизнес-процессов, настройки, информация по текущим исполняемым функциям и другие данные периодически сохраняются в долговременной памяти (в базе данных). Таким образом, даже если программа-распорядитель выполнения работ завершит в силу непредвиденных обстоятельств свою нормальную работу, то при следующем запуске последнее стабильное рабочее состояние будет восстановлено.

### ***3.2. Краткий обзор архитектуры Shark***

На примере архитектуры системы Shark можно продемонстрировать большинство современных подходов к декомпозиции и методам реализации сложной объектно-ориентированной программной системы. Прежде всего, система Shark использует принцип «клиент-сервер». Алгоритмы управления и выполнения бизнес-процессов, механизмы вызова внешних систем сконцентрированы в одном приложении — сервере («Back End»). Причем на сервере намеренно отсутствует реализация какого-либо пользовательского интерфейса. Сервер предоставляет к использованию набор открытых интерфейсов. Эти интерфейсы используются для вызова различных функций (методов) сервера

многочисленными независимыми приложениями — клиентами («Front End»). Реализация клиентов, в отличие от сервера, содержит алгоритмы и структуры данных, выполняющие лишь визуализацию и прием информации от пользователя. В качестве клиентов могут выступать различные программные системы других производителей.

В системе Shark реализовано два типа интерфейсов для взаимодействия клиентской и серверной части. Первый тип интерфейсов — это набор обычных Java-интерфейсов и абстрактных классов. При использовании этого типа интерфейсов разделение сервера и клиента происходит лишь на логическом уровне. Реально клиентская и серверная часть составляют единое приложение, работающее на одной виртуальной машине.

Второй тип интерфейсов построен на основе использования широко распространенной технологии CORBA (Common Object Requests Broker). Эта технология, разрабатываемая известным консорциумом OMG, позволяет просто вызывать методы удаленных объектов. Клиент и сервер в этом случае являются двумя независимыми приложениями, они могут располагаться на различных компьютерах и взаимодействовать по сети.

Серверная часть системы Shark разделена на три горизонтальных уровня (рис. 3.1):

- 1) уровень внешних интерфейсов;
- 2) уровень каркаса;
- 3) уровень внутренних служб.

На первом уровне внешние интерфейсы сгруппированы по функциональному признаку. Каждая группа интерфейсов обеспечивает выполнение определенной узкой задачи (работа с хранилищем определений процессов, управление активными экземплярами процессов в оперативной памяти сервера и т.д.). Каждый интерфейс имеет соответствующую реализацию в виде конкретного класса, который непосредственно взаимодействует со стабильной частью системы — каркасом. На уровне каркаса реализованы основные алгоритмы распорядителя выполнением работ (workflow engine): работа с определениями и экземплярами бизнес-процессов, воплощена логика выполнения функций, срабатывания пе-

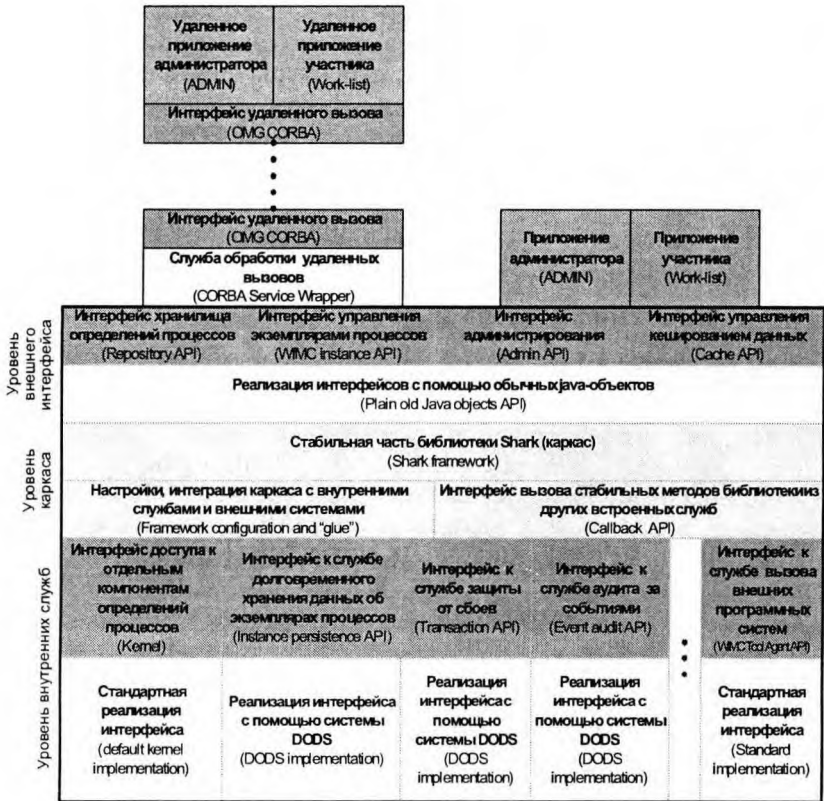


Рис. 3.1. Архитектура системы Shark

реходов, передачи данных, вызова приложений и пр. Стабильность реализации распорядителя выполнением работ на уровне каркаса обеспечивается за счет того, что в этих алгоритмах сущности, обозначающие процессы, функции и другие понятия бизнес-процессов, выражены в терминах абстрактных интерфейсов. Реализация интерфейсов, а также всех вспомогательных алгоритмов, которые нестабильны и могут изменяться в зависимости от технологий, конфигурации системы и прочих факторов, вынесена в третий уровень внутренних служб. Доступ к этому уровню осуществляется через методы абстрактных интерфейсов.

На уровне внутренних служб наиболее активно используются возможности по динамической загрузке и позднему связыванию java-классов. В файле конфигурации системы Shark явно задано соответствие абстрактного интерфейса реальному классу, ответственному за выполнение определенной группы задач. Поэтому разработчики и пользователи системы могут просто расширять или адаптировать возможности внутренних служб путем создания собственных классов, применения других системных библиотек и изменения конфигурационного файла системы.

Чаще всего подвергаются изменению служба вызова внешних программных систем для исполнения функций бизнес-процесса и служба долговременного хранения данных. Именно эти службы наиболее сильно связаны со спецификой конкретной организации, активно взаимодействуя с базами данных, внешними программными системами и пользователями. Поэтому они постоянно расширяются путем добавления новых классов, в которых реализованы специфические алгоритмы.

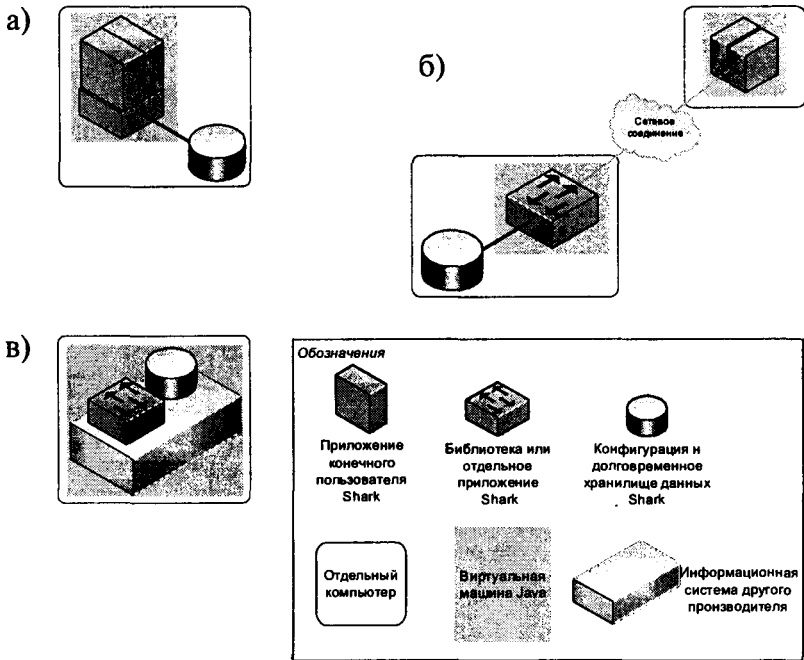
На рис. 3.1 представлена лишь часть внутренних служб, полный их перечень приведен в табл. 3.1.

За счет гибкости и расширяемости архитектуры система Shark может эксплуатироваться в трех различных вариантах (рис. 3.2). Причем различные копии сервера Shark могут использовать одно и то же хранилище данных. В этом случае копии дублируют работу друг друга, а общая надежность системы повышается.

**Внутренние службы системы Shark***Таблица 3.1*

<b>№</b>	<b>Русское обозначение</b>	<b>Английское обозначение</b>
1	Интерфейс доступа к отдельным компонентам определений процессов	Kernel
2	Интерфейс к службе долговременного хранения данных об экземплярах процессов	Instance persistence API
3	Интерфейс к службе защиты от внутренних сбоев	Transaction API
4	Интерфейс к службе аудита за событиями	Event audit API
5	Интерфейс к службе вызова внешних программных систем для исполнения функций	WfMC Tool Agent API
6	Интерфейс к службе назначения участников бизнес-процесса реальным пользователям системы Shark	Asignme API
7	Интерфейс к службе проверки прав доступа к Shark	Authentication API
8	Интерфейс к службе защиты от сбоев в работе внешних программных систем	User/Authentication transaction
9	Интерфейс к службе долговременного хранения данных о пользователях системы Shark	User/Group persistence
10	Интерфейс к службе блокировки определения процесса	Process locking
11	Интерфейс к службе управления различными интерпретаторами	Script API
12	Интерфейс к службе ведения протокола работы системы Shark	Logging API
13	Интерфейс к службе кеширования данных	Caching API
14	Интерфейс к службе ресурсных ограничений для приложений	Limit Agent API
15	Интерфейс к службе передачи параметров вызова (имя, пароль пользователя) внешним программным системам	Security API
16	Интерфейс к службе интеграции с другими системами workflow management	Integration API
17	Интерфейс к службе долговременного хранения правил соответствия приложений бизнес-процесса реальным программным системам	Application Map persistence
18	Интерфейс к службе долговременного хранения правил соответствия выражений интерпретаторам, их обрабатывающим	Script Map persistence
19	Интерфейс к службе долговременного хранения правил соответствия участников бизнес-процесса пользователям системы Shark	Participant Map persistence
20	Интерфейс к службе долговременного хранения описаний бизнес-процессов	Repository persistence





**Рис. 3.2.** Различные варианты использования системы Shark: а — как единое приложение; б — распределенная система «клиент-сервер»; в — «встроенный» компонент в составе другой программной системы

### 3.3. Пример настройки и работы Shark

#### 3.3.1. Установка продукта

Инсталлятор Shark для любой версии может быть скачан бесплатно с сайта консорциума ObjectWeb:

[http://forge.objectweb.org/project/showfiles.php?group\\_id=74](http://forge.objectweb.org/project/showfiles.php?group_id=74).

Существуют версии для различных операционных систем.

Версия Shark 1.0-1 требует наличия JDK 1.4.1 или старше.

### 3.3.2. Структура каталогов и основные программы администрирования

Система Shark имеет достаточно глубокую структуру каталогов. На рис. 3.3 представлены только наиболее важные для пользователя и администратора каталоги двух первых уровней.

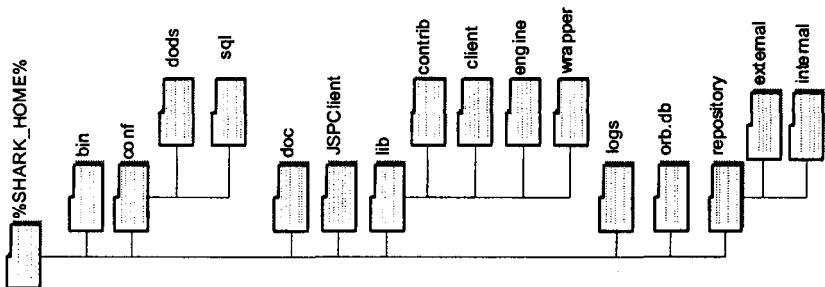


Рис. 3.3. Структура основных каталогов системы Shark

Все пользовательские программы и сервер системы Shark запускаются скриптами соответствующей операционной системы. Скрипты расположены в каталоге `%SHARK_HOME%\bin` (табл. 3.2).

### 3.3.3. Примеры работы с администраторским приложением Shark

Рассмотрим, как происходит работа администраторов или пользователей системы Shark. В качестве первого примера возьмем описание простого бизнес-процесса по обработке заказов, поставляемого в составе Shark, в котором все функции выполняются вручную.

#### *Простой учебный пример*

В начале работы необходимо запустить систему Shark в режиме самостоятельного приложения (CORBA-сервер). Так как нам необходимо сразу приступить к администриро-

## Основные сущности мета-модели языка XPDL

Приложение	Назначение
Chainsaw	Запуск вспомогательной графической программы для удобного просмотра и анализа протокола работы системы Shark (Log Viewer)
recreateDB	Запуск процедуры полной очистки долговременного хранилища (базы данных) с определениями и состоянием запущенных на выполнение экземпляров бизнес-процессов. Приложение работает в консольном режиме
run	Запуск Shark в виде самостоятельного приложения (CORBA-сервер). Приложение работает в консольном режиме
runA	Запуск графического приложения для удаленного администрирования Shark (CORBA-клиент)
runAll	Одновременный запуск Shark в виде самостоятельного приложения (CORBA-сервер) и графического приложения для удаленного администрирования Shark (CORBA-клиент)
runCPS	Запуск вспомогательной консольной программы, которая запускает на исполнение экземпляр на удаленной системе Shark
runSA	Запуск графического приложения для работы с возможностями Shark в режиме локальной библиотеки
runts	Запуск различных вариантов тестов для локального и удаленного режима работы Shark
runWH	Запуск графического приложения для удаленной работы с назначенными на исполнение функциями бизнес-процесса (CORBA-Worklist Handler)
sharkCorbaService Install	Инсталляция Shark в виде сервиса ОС Windows. Приложение работает в консольном режиме
sharkCorbaService Start	Запуск приложения Shark как сервис ОС Windows
sharkCorbaService Stop	Остановка приложения Shark, запущенного ранее как сервис ОС Windows. Приложение работает в консольном режиме
sharkCorbaService Uninstall	Деинсталляция Shark, инсталлированного ранее в виде сервиса ОС Windows. Приложение работает в консольном режиме
tns	Запуск служебного процесса (Name Service) для удаленного доступа к Shark с помощью интерфейса CORBA. Приложение работает в консольном режиме

ванию, имеет смысл запустить вместе с системой Shark и графическое администраторское приложение. Поэтому для запуска воспользуемся программой

`%SHARK_HOME%\bin\runAll.`

В результате работы этой программы будет запущена система Shark, а также появится графическое окно администраторского приложения (рис. 3.4).

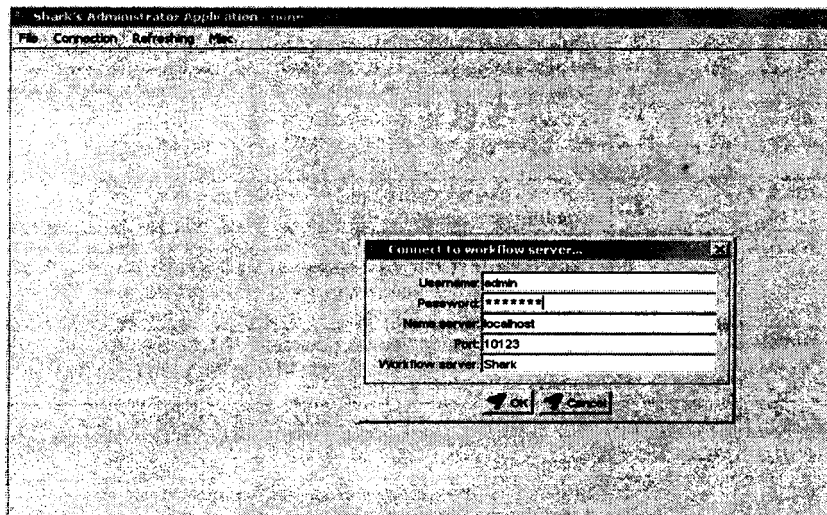


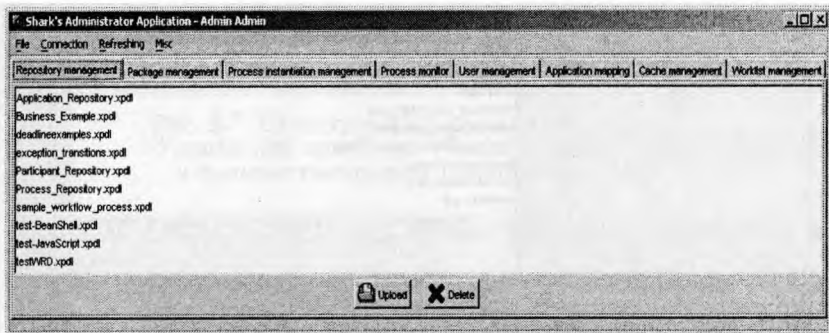
Рис. 3.4. Начало работы с администраторским приложением Shark (вариант CORBA-клиента)

Для начала работы в нем нужно ввести параметры соединения. Обычно требуется ввод имени пользователя и его пароля (поля «Username» и «Password»). По умолчанию система Shark в режиме самостоятельного приложения (CORBA-сервер) содержит информацию об одном пользователе и требует ввода таких значений:

- ◆ Username: admin;
- ◆ Password: enhydra.

Отметим, что в ходе работы в административном приложении можно будет добавлять новых пользователей и изменять пароли уже существующих.

После успешного ввода имени, пароля и регистрации в системе графический интерфейс администраторского приложения выглядит следующим образом (рис. 3.5)



**Рис. 3.5.** Общий вид интерфейса администраторского приложения Shark

Интерфейс имеет достаточно простую структуру — главное окно приложения содержит основное меню и восемь закладок:

- ◆ Repository management — управление описаниями бизнес-процессов;
- ◆ Package management — управление пакетами;
- ◆ Process instantiation management — управление экземплярами процессов;
- ◆ Process monitor — мониторинг процессов;
- ◆ User management — управление пользователями системы Shark;
- ◆ Application mapping — отображение программных систем на приложения бизнес-процессов;
- ◆ Cache management — управление кэшированием данных в системе Shark;
- ◆ Worklist management — управление назначенными для исполнения функциями.

На каждой закладке находится специфический интерфейс, предназначенный для выполнения определенного множества задач по администрированию системы Shark или управлению бизнес-процессами, пользователями, приложениями и т.п.

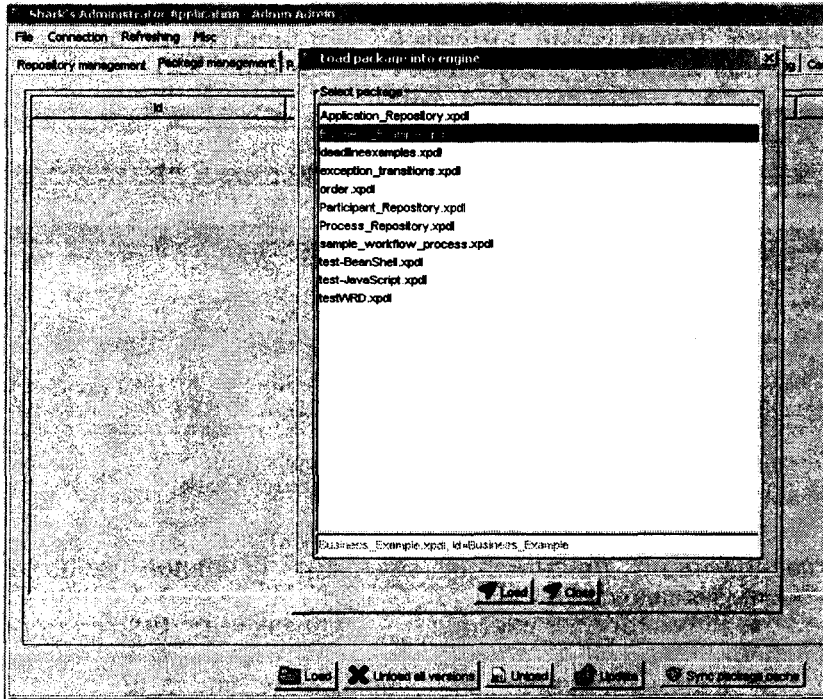


Рис. 3.6. Загрузка новых пакетов в распорядитель работ

Сразу после успешной регистрации и подключению к Shark в администраторском приложении становится активной закладка «Управление описаниями бизнес-процессов» («Repository management»). В ней перечислены все доступные для Shark и находящиеся в долговременном хранилище описания процессов в виде текстовых XPD-файлов. На этой же закладке можно выполнить загрузку нового XPD-файла в хранилище системы Shark или удалить XPD-файл из системы. Так как мы используем в примере описание бизнес-процесса, поставляемое вместе с системой (Business\_Example.xpd), то никаких действий здесь производить не нужно, а можно переходить к закладке «Управление пакетами» (Package management) (рис. 3.6).

Shark's Administrator Application - Admin Admin

Connection Refreshing Disabled

User management    Application mapping    Cache management    Version management

Repository management    Package management    Process installation management    Process monitor

Package	Version	Name	No. of process definitions
Application_Repository	2	Application Repository	0
Business_Example	2	Business Example	3
Participant_Repository	2	Participant Repository	0
Process_Repository	2	Process Repository	2

**Рис. 3.7.** Структура информации на закладке «Управление пакетами» (Package management) в администраторском приложении Shark

Shark's Administrator Application - Admin Admin

Connection Refreshing Disabled

User management    Application mapping    Cache management    Version management

Repository management    Package management    Process installation management    Process monitor

Engine context

- [-] General\_Repository
  - [-] Package-Business\_Example
    - Process definition-Sales Order Processing
    - Process definition-Customer Service - Request for
    - Process definition-Check Credit
  - [-] Package-Participant\_Repository
  - [-] Package-Process\_Repository
  - [-] Package-Application\_Repository

Process properties

Property	Value
Name	
Category	
Version	
Active processes	
State	

Instantiate    View    Description    Enable    Disable    Recalculate assignments

**Рис. 3.8.** Схема Структура интерфейса на закладке «Управление экземплярами процессов» (Process instantiation management) в администраторском приложении Shark

Мы уже знаем, что описания процессов, данных, участников и приложений группируются по пакетам. При этом один XPDL-файл может содержать несколько пакетов. Для того, чтобы запустить процесс на выполнение, необходимо загрузить соответствующий пакет (и, возможно, еще несколько связанных с ним) из долговременного хранилища в оперативную память системы Shark. В администраторском приложении загрузка пакетов из файла производится по нажатию кнопки «Load». В появившемся диалоговом окне

нужно выбрать соответствующий файл и нажать на кнопку. Вслед за этим происходит загрузка всех пакетов, содержащихся в файле, и информация по пакетам отображается на закладке (рис. 3.7). Каждый из пакетов может быть выделен и выгружен из оперативной памяти, либо синхронизирован с находящимся в хранилище вариантом.

После загрузки пакетов из файла `Business_Example.xpdl` можно выполнить операцию по созданию в системе Shark активного экземпляра бизнес-процесса из пакета «`Business_Example`». Для этого служит закладка «Управление экземплярами процессов» (`Process instantiation management`) (рис. 3.8).

В левой части интерфейса на этой закладке находится дерево пакетов и определенных в них процессов. Раскрытие узлов дерева производится двойным нажатием левой клавиши мыши. В начальный момент дерево пакетов скрыто (отображается только корень — «`Opened packages`»). На рис. 3.8 видно, что пакет «`Business_Example`» содержит три описания процессов. Выделение какого-нибудь описания (например «`Process definition — Sales Order Processing`») приводит к отображению в правой части общих сведений и информации по активным экземплярам этого процесса (рис. 3.9). Для выделенного описания можно посмотреть текстовую информацию (кнопка «`Description`»), а также получить представление процесса в форме графа, которая используется в редакторе JaWE (кнопка «`View`»). В рассматриваемом примере мы создадим экземпляр процесса «`Sales Order Processing`». Выделяем описание этого процесса в дереве и нажимаем кнопку «`Instantiate`». При этом в правой области окна (в поле «`Active processes`») значение увеличивается на единицу.

Работа с экземплярами запущенных процессов происходит на закладке «Мониторинг процессов» (`Process monitor`) (рис. 3.10).

Интерфейс на этой закладке разделен на четыре области. В левой верхней области отображается уже знакомое вам дерево пакетов и определений процессов. Отличительной особенностью этого дерева является то, что узлы определений могут иметь подчиненные узлы (обозначаются флажком). Они соответствуют созданным экземплярам од-



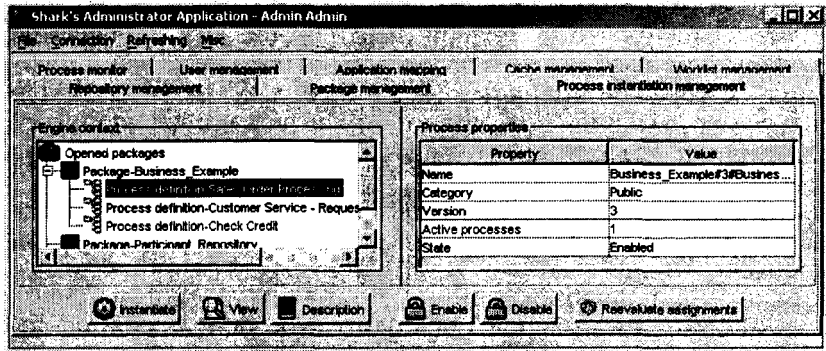


Рис. 3.9. Доступная информация по процессу на закладке «Управление пакетами» (Package Management) в администраторском приложении Shark

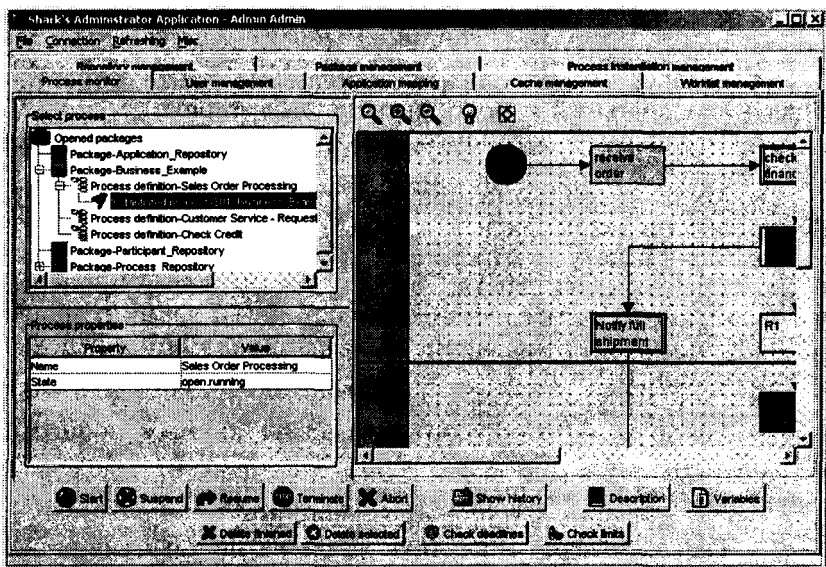


Рис. 3.10. Структура интерфейса на закладке «Мониторинг процессов» (Process monitor) в администраторском приложении Shark

ного и того же процесса. Выделение узла-экземпляра приводит к тому, что в правой верхней области появляется графическое изображение структуры этого процесса, а функция, которая выполняется в данный момент, выделяется (например, на рис. 3.10 видно, что в экземпляре процесса «Sales Order Processing» сейчас выполняется функция «receive order»).

В левой области отображается подробная информация по выделенному экземпляру, из которой можно узнать имя процесса (поле «Name») и его текущее состояние (поле «State»). За подробностями по принятым в системе Shark состояниям экземпляров обращайтесь к оригинальной документации.

Нижняя область закладки содержит кнопки, с помощью которых можно выполнить различные действия по управлению ходом исполнения экземпляров процессов:

- ◆ Start — запустить экземпляр, если он находится в состоянии open.not\_run-ning.not\_started;

- ◆ Suspend — приостановить все исполняющиеся функции и подпроцессы;

- ◆ Resume — продолжить исполнение ранее приостановленных функций и подпроцессов экземпляра;

- ◆ Terminate — завершить работу всех функций и подпроцессов экземпляра;

- ◆ Abort — аварийно прервать работу всех функций и подпроцессов экземпляра;

- ◆ Show history — отобразить хронологический отчет по всем событиям во время работы экземпляра процесса (начало работы, время изменения состояния, время изменения элементов данных, время изменения состояния отдельной функции и т.п.);

- ◆ Description — отобразить описание процесса;

- ◆ Variables — отобразить элементы данных процесса для просмотра или редактирования;

- ◆ Activity management — открыть дополнительный диалог для управления исполнением функций;

- ◆ Delete finished — удалить из оперативной памяти все завершенные на данный момент экземпляры процесса;

- ◆ Delete selected — удалить из оперативной памяти все выделенные в дереве экземпляры процесса;

◆ Check deadlines — проверить ограничение на крайний срок выполнения всех функций;

◆ Check limits — проверить ограничение на крайний срок выполнения всех функций и процессов.

В нашем простейшем примере выбран такой бизнес-процесс, в котором все функции выполняются исполнителями в ручном режиме. Поэтому один пользователь шаг за шагом может исполнить все функции этого бизнес-процесса. Но, как вы знаете, перед началом выполнения бизнес-процесса необходимо поставить в соответствие абстрактным участникам бизнес-процесса конкретных пользователей системы (в данном случае — пользователей системы Shark). Обычно это делается на закладке «Управление пользователями системы Shark» («User management»).

Мы воспользуемся особенностью реализации системы Shark — если для абстрактного участника не поставлено в соответствие ни одного реального пользователя системы Shark, то на исполнение функции автоматически назначается пользователь с именем «admin» — администратор системы. Так как в административном приложении мы уже зарегистрированы с этим именем, то можно сразу же переходить на закладку «Управление назначенными для исполнения функциями» (Worklist management). Интерфейс, реализованный на ней, позволяет принимать на исполнение назначенные функции, изменять переменные процесса и завершать исполнение функции (рис. 3.11).

Из рис. 3.11 видно, что на закладке находится список всех функций (возможно относящихся к различным экземплярам и даже к различным процессам), назначенных выбранному пользователю. Самым первым шагом при ручном исполнении определенной функции является подтверждение начала ее исполнения. Для этого необходимо сделать выбор в поле «Accepted» на строчке, соответствующей исполняемой функции. Лишь после выбора функция будет считаться принятой к исполнению, и распорядитель работ будет вести статистику по времени ее исполнения (поля «Started» и «Duration»).

Особенностью системы Shark является то, что она не только позволяет отследить начало и конец исполнения ручных функций. Пользователи также могут просматривать



**Рис. 3.11.** Структура интерфейса на закладке «Управление назначенными для исполнения функциями» (Worklist management) в администраторском приложении Shark

и изменять данные процесса, которые с этой функцией связаны. Такая возможность не является стандартной и не определяется строго в рекомендациях WfMC. Поэтому использование этой возможности ограничено рамками системы Shark — эта система понимает особый набор расширенных атрибутов, которые можно включать в описание любой функции бизнес-процесса на языке XPDL. Эти особые расширенные атрибуты определяют, какие данные процесса при выполнении функции доступны для изменения или просмотра. За подробностями по этим расширенным атрибутам следует дождаться рассмотрения второго примера или обратиться к оригинальной документации.

Итак, если при определении функции в XPDL-файле были добавлены особые расширенные атрибуты, перечисляющие элементы данных процесса, связанные с этой функцией, то пользователь во время исполнения функции может их изменить или посмотреть, нажав на кнопку «Update variable(s)» (рис. 3.12).

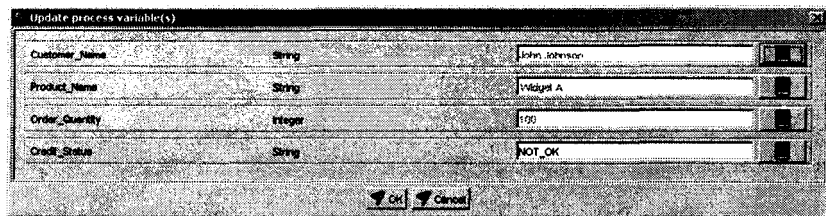


**Рис. 3.12.** Диалог для модификации элементов данных процесса во время исполнения назначенной функции в администраторском приложении Shark (все данные доступны для редактирования)

В данный момент выполняется функция «receive order» и в диалоговом окне «Update process variables(s)» можно ввести все ключевые данные для обработки (имя заказчика, название товара, количество). После модификации разрешенных к изменению элементов данных процесса можно закрыть диалог и уведомить распорядитель работ о завершении ручного исполнения функции. Для этого служит кнопка «Complete» (рис. 3.11).

После этого в соответствии с описанием бизнес-процесса распорядитель работ запускает на исполнение следующую функцию «check credit». Она также в нашем примере выполняется в ручном режиме, поэтому на закладке «Управление назначенными для исполнения функциями» (Worklist management) спустя некоторое время появляется новый элемент в списке назначенных функций. Необходимо подтвердить начало выполнения новой функции, опять выбрав поле «Accepted». Если теперь нажать на кнопку «Update process variables(s)», то можно увидеть, что часть данных процесса доступна только для чтения (рис. 3.13).

Единственным разрешенным для изменения данным является статус кредита для заказчика. После просмотра или редактирования данных при исполнении функции «check credit» не забудьте уведомить распорядитель работ о том, что ручное исполнение функции завершено (нажмите на кнопку «Complete»). Если во время исполнения функции «check credit» не изменять элемент данных «Credit\_Status», то это будет означать, что функция закончилась с отрицательным результатом и в соответствии с описанием бизнес-



**Рис. 3.13.** Диалог для модификации элементов данных процесса во время исполнения назначенной функции в администраторском приложении Shark (часть данных доступна только для просмотра)

процесса будет запущена на исполнение функция «decline order». В ней никаких элементов данных для редактирования нет, поэтому после подтверждения начала исполнения можно сразу нажать на кнопку «Complete».

Функция «decline order» является последней функцией бизнес-процесса, поэтому активный экземпляр процесса завершает свою работу — на закладке меняется состояние этого экземпляра (State = closed.completed).

На этом пример можно считать завершенным, необходимо из администраторского приложения «выключить» систему Shark (команда основного меню «Connection/Shutdown engine») и выйти из программы (команда основного меню «File/Exit»).

*Пример  
работы с определением бизнес-процесса,  
созданным в гл. 2*

Рассмотрим теперь более реальный и сложный случай, когда в систему Shark нужно загрузить внешний XPDL-файл, настроить отображение исполнителей на пользователей системы и сконфигурировать вызов программных систем как приложений бизнес-процесса. Для этого примера мы возьмем описание бизнес-процесса, созданного в редакторе JaWE на прошлой лекции. Кроме администраторского приложения системы Shark на этот раз нам также понадобится редактор JaWE и среда разработки Eclipse. Для определенности мы будем считать, что описание бизнес-процесса находится в файле order.xpdl и размещается в том же каталоге, где установлен редактор JaWE.

Как и в предыдущем примере, запустим одновременно систему Shark в режиме самостоятельного приложения и администраторское приложение, вызвав

```
%SHARK_HOME%\ bin\runAll.
```

После регистрации в администраторском приложении попытаемся загрузить определения, находящиеся в файле order.xpdl в долговременное хранилище системы Shark. Для этого на закладке «Управление описаниями бизнес-процессов» («Repository management») нажмем кнопку «Load». Появится стандартный диалог открытия файлов, из которого доступна локальная файловая система — т.е. файлы на том компьютере, где работает администраторское приложение<sup>12</sup>.

В диалоге нужно выбрать файл

```
%JAWE_HOME%\ order.xpdl.
```

Перед загрузкой файла в систему Shark будет проведена тщательная проверка на его корректность и соответствие всем ограничениям. По результатам проверки на экран будет выдан отчет, в той же форме, что и отчет на правильность в редакторе JaWE. На этом этапе нас ждет сюрприз — так как правила проверки, реализованные в Shark, значительно строже, а, кроме того, в самом распорядителе работ не полностью реализованы возможности языка XPDЛ, то корректное с точки зрения JaWE описание будет для Shark признано ошибочным (рис. 3.14).

На этом рисунке нами все строки пронумерованы, а все важные сообщения об ошибках выделены рамкой. Рассмотрим сначала причину этих сообщений, а потом кратко объясним, что нужно изменить в описании бизнес-процесса, чтобы исправить ошибки.

### Ошибка № 1

Строки 3—6. Ошибка в описании формальных параметров подпроцесса Check. В атрибуте «Index» вместо

---

<sup>12</sup> На это не случайно обращается ваше внимание. В общем случае система Shark и администраторское приложение могут работать на разных компьютерах. Поэтому те новые XPDЛ-файлы, которые нужно загрузить, должны быть доступны именно на компьютере, где работает администраторское приложение.

1. **XPDL schema**
2. **F:\Shark-1.0\repository\external\order\_o.xpdl**
3. **[Error] at line number 288: cvc-attribute.3: The value 'Order Status' of attribute 'Index' on element 'FormalParameter' is not valid with respect to its type, 'NMTOKEN'.**
4. **[Error] at line number 283: cvc-datatype-valid.1.2.1: 'Order Amount' is not a valid value for NMTOKEN.**
5. **[Error] at line number 283: cvc-attribute.3: The value 'Order Amount' of attribute 'Index' on element 'FormalParameter' is not valid with respect to its type, 'NMTOKEN'.**
6. **[Error] at line number 288: cvc-datatype-valid.1.2.1: 'Order Status' is not a valid value for 'NMTOKEN'.**
7. **Package:order**
8. **Connections**
9. All elements are properly connected !!!
10. **Graph conformance**
11. The graphs of all processes within the package conforms to the given graph conformance class !!!
12. **Logic**
13. **Process:** Id= 1 ,Name= Process
14. *The process contains one or more logic errors*
15. **Process:** Id= Check, Name= Check
16. *The process contains one or more logic errors*
17. **Type declaration:** Id= Order\_Status\_Type, Name= Order Status Type
18. *The type declaration is not supported - currently supported type declarations are for basic types: BOOLEAN, STRING, INTEGER, FLOAT, DATETIME, as well as for their type declaration versions*
19. **Workflow process:****Process**
20. **Connections**  
All elements are properly connected !!!
21. **Graph conformance**
22. Process graph conforms to the given graph conformance class !!!

**Рис. 3.14.** Отчет по результатам проверки исходного определения *order.xpdl*



23. Logic
24. **Generic:** Id= l\_Act1 ,Name= ReceiveOrder
25. *'AUTOMATIC' start mode not allowed for 'Tool' activities with performers other than 'System' type participant or empty expression*
26. **Generic:** Id= Decline, Name= Decline
27. *'AUTOMATIC' start mode not allowed for 'Tool' activities with performers other than 'System' type participant or empty expression*
28. **Generic:** Id= Finish\_Order, Name= Finish\_Order
29. *'AUTOMATIC' start mode not allowed for 'Tool' activities with performers other than 'System' type participant or empty expression*
30. **Workflow relevant data:** Id= Order\_Status, Name= Order Status
31. *Data type is not supported - currently supported types are basic types: BOOLEAN, STRING, INTEGER, FLOAT, DATETIME, as well as their type declaration versions*
32. Workflow process:Check
33. Connections
34. All elements are properly connected !!!
35. Graph conformance
36. Process graph conforms to the given graph conformance class !!!
37. Logic
38. **Generic:** Id= Check, Name= Check
39. *'AUTOMATIC' start mode not allowed for 'Tool' activities with performers other than 'System' type participant or empty expression*
40. **Formal parameter:** Id= status,
41. *Data type is not supported - currently supported types are basic types: BOOLEAN, STRING, INTEGER, FLOAT, DATETIME, as well as their type declaration versions*
42. **Formal parameter:** Id= Order\_Status,
43. *Data type is not supported - currently supported types are basic types: BOOLEAN, STRING, INTEGER, FLOAT, DATETIME, as well as their type declaration versions*

Рис. 3.15. Отчет по результатам проверки исходного определения *order.xpdl* (окончание)

порядкового номера (0,1,2 и т.д.) было указано само символическое имя параметра.

### Ошибка № 2

Строки 17—18, 40—43. Ошибка произошла из-за того, что в текущей версии системы Shark не поддерживаются сложные типы данных.

### Ошибка № 3

Строки 24—129. В системе Shark запрещено автоматически запускать на исполнение функцию и приложение даже в том случае, если на ее исполнение назначен участник с типом, отличным от «System». Действительно, если на исполнение назначен участник с типом «Role», «Human» или «Organizational Unit», то, скорее всего, реально в исполнении функции будет задействован определенный пользователь системы Shark. Он должен иметь возможность вручную подтвердить начало исполнения функции, а после этого начнется работа приложения. Лишь в том случае, когда на исполнение функции назначен участник типа «System», исполнение функции находится под контролем распорядителя работ от начала и до конца и имеет смысл автоматически начинать ее исполнение.

Рассмотрим возможные способы разрешения перечисленных ошибок. Все исправления, естественно, следует выполнять в редакторе JaWE, а исправленное определение записать в файле с другим именем, например, `order_fixed.xpdl`.

◆ Ошибка № 1 разрешается путем изменения значения атрибута «Index» для каждого формального параметра подпроцесса «Check». Вместо текстового значения нужно использовать порядковые числа, начиная с 0 для первого формального параметра «Customer\_Account». За справкой по необходимым при этом действиям в редакторе JaWE обратитесь к шагу № 7 гл. 2. Убедитесь также, что значения атрибута «Index» у формальных аргументов всех приложений также являются порядковыми целыми числами.

◆ Ошибка № 2 разрешается путем изменения определения пользовательского типа данных «Order\_Status\_Type». Вместо того, чтобы быть перечислимым типом, он должен стать разновидностью стандартного типа «String». Чтобы вспомнить правила определения и редактирования пользо-

вательских типов данных в редакторе JaWE, рассмотрите шаг № 6 гл. 2. После редактирования типа данных «Order\_Status\_Type» потребуется также внести изменения в условия срабатывания перехода от функции «CheckAccount» к функции «Finish\_Order». Изменение небольшое, но важное — в правильной версии описания бизнес-процесса условие срабатывания этого перехода должно быть таким: `Order_Status == «OK»` (по сравнению с исходной версией константа OK заключена в кавычки).

◆ Ошибка № 3 потребует изменения значения атрибута «Start mode» в определении всех функций, где для исполнения используются приложения. Вместо значения «Automatic» в таких функциях нужно использовать значение «Manual». Используя материалы прошлой главы (шаг № 4), измените значение атрибута «Start mode» в функциях «Receive\_Order», «Decline» и «Finish\_Order» главного процесса, а в функции «Check», определенной в подпроцессе «Check», кроме того, удалите связь с инструментальным средством. Пусть эта функция будет исполняться в ручном режиме.

Если после внесения всех исправлений попробовать загрузить файл `order_fixed.xpdl` в систему Shark, то никаких предупреждений об ошибках быть не должно. Поэтому можно загрузить определения пакетов из этого файла. Однако не спешите сразу это делать и запускать экземпляр процесса «order» — перед этим нам необходимо выполнить ряд важных действий по определению доступных в функциях элементов данных, созданию реальных программ, исполняющих роль приложений, и определению соответствий между приложениями и программами.

Как уже указывалось ранее, к определению функции в XPDL-файле нужно добавить расширенные атрибуты. После этого система Shark сможет обеспечить редактирование указанных элементов данных либо пользователем, ответственным за исполнение функции, либо приложением.

Добавлять расширенные атрибуты к определению функции лучше всего в редакторе JaWE. При этом нужно руководствоваться следующими правилами:

◆ если желательно, чтобы во время исполнения функции элемент данных процесса с именем «x1» был доступен

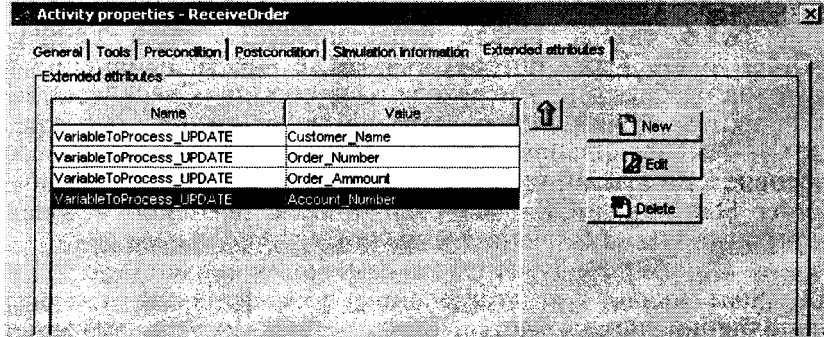


Рис. 3.16. JaWE — описание функции с добавленными расширенными атрибутами

для изменения, то необходимо включить в описание функции расширенный атрибут с именем «VariableToProcess\_UPDATE», значение этого атрибута должно быть «x1»;

◆ если желательно, чтобы во время исполнения функции элемент данных процесса с именем «x2» был доступен только для чтения, то необходимо включить в описание функции расширенный атрибут с именем «VariableToProcess\_VIEW», значение этого атрибута должно быть «x2».

На рис. 3.16 показано окно редактора JaWE с набором расширенных атрибутов для функции «ReceiveOrder». Для других функций выполните добавление самостоятельно, руководствуясь соображениями здравого смысла.

Похожим образом нужно добавить расширенные атрибуты для функции «Check» в подпроцессе «Check». Только теперь в качестве значений расширенных атрибутов «VariableToProcess\_UPDATE» или «VariableToProcess\_VIEW» должны выступать имена формальных параметров этого подпроцесса.

Обратим ваше внимание еще на одно изменение в исходном описании бизнес-процесса. Во время работы с определением процесса в системе Shark выяснилось также, что нужно строго следить за правильным порядком следования исходящих переходов в определении функций. Например, в первоначальном варианте для функции

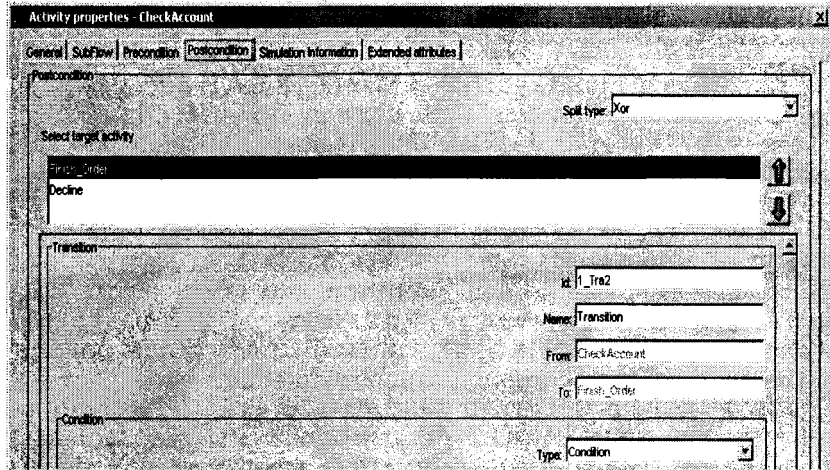


Рис. 3.17. JaWE — правильный порядок следования выходных переходов в функции «Check»

«check\_credit» в начале идет выходной переход с типом «OTHERWISE». Если оставить такой порядок без изменения, то всегда будет срабатывать именно этот переход и, соответственно, исполняться функция «decline». Поэтому порядок нужно поменять так, чтобы в начале списка переходов стоял переход на функцию «Finish\_Order» с «CONDITION» (рис. 3.16).

После сохранения всех сделанных изменений в файле под именем order\_full.xpdl и загрузки его в хранилище системы Shark можно переходить к разработке программ, играющих роль приложений при исполнении бизнес-процесса. В основной части мы рассмотрим лишь правила создания программ в виде java-классов с использованием Eclipse. Другие варианты (отправка почты, вызов стандартных внешних программ, использование скриптовых языков и т.д.) кратко рассмотрены в приложении, а подробно — в оригинальной документации.

Согласно определению бизнес-процесса в нем используется три различных приложения. Их сигнатуры на псевдокоде представлены далее:

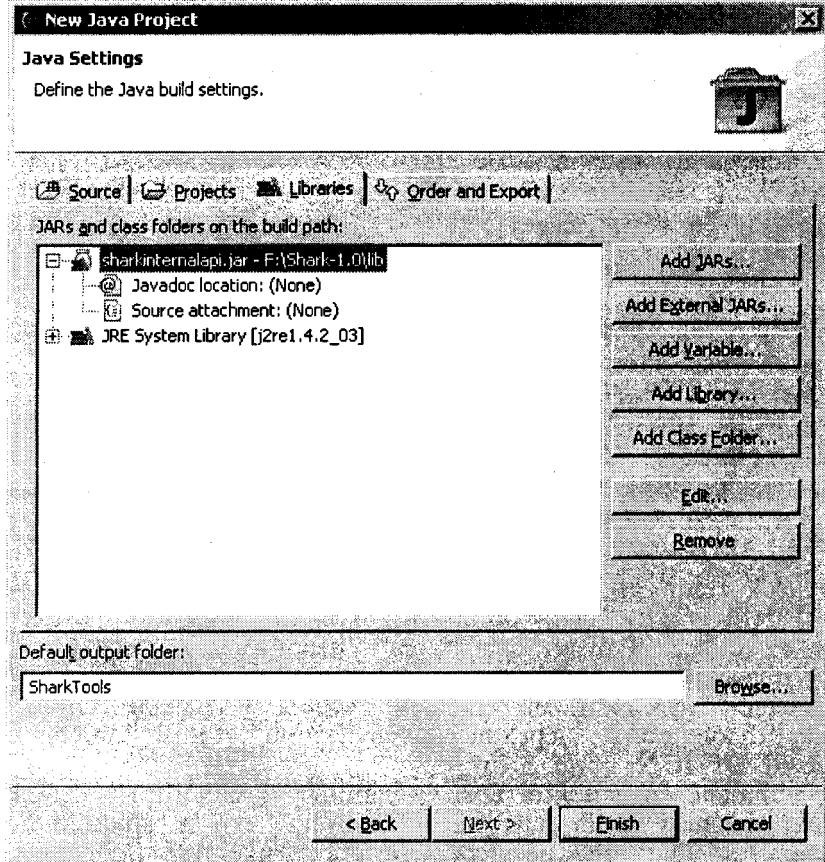


Рис. 3.18. Подключение дополнительных библиотек в проект

- ◆ getOrderDetails (OUT String cust\_name, OUT String cust\_account, OUT Integer order\_id, OUT Float order\_amount);
- ◆ doFinish(IN String cust\_name, IN Integer order\_id);
- ◆ doDecline(IN String cust\_name, IN Integer order\_id).

Рассмотрим, как происходит процесс создания трех классов на языке Java, которые соответствуют этим приложениям и могут быть вызваны из системы Shark. Прежде

всего, необходимо создать новый проект в среде разработки Eclipse и в настройках этого проекта подключить внешний jar-файл `%SHARK_HOME%\lib\sharkinternalapi.jar` (`%SHARK_HOME%` должен быть заменен реальным путем, где установлена система) (рис. 3.17).

При разработке классов полной инсталляции системы Shark не потребуется.

Затем в проекте Eclipse следует создать новый пакет (java-package), в котором будут размещаться три класса, соответствующие приложениям бизнес-процесса. Для определенности этот пакет будет носить название `edu.hse.nnov.bi.shark`. Наконец, внутри этого пакета можно создать класс, соответствующий приложению `GetOrderDetails`. Это должен быть обязательно публичный класс с произвольным именем, например, `SharkOrderDetails`.

По требованиям системы Shark для того, чтобы публичный java-класс мог в дальнейшем использоваться при исполнении функции бизнес-процесса, в этом классе должен быть реализован публичный статический метод `execute`. В этом методе число, порядок и имена формальных параметров должны быть такими же, что и у соответствующего приложения в определении процесса. Тип всех параметров метода `execute` один и тот же. Это служебный тип `org.enhydra.shark.api.internal.toolagent.AppParameter`, определенный в jar-файле `%SHARK_HOME%\lib\sharkinternalapi.jar`. Определение этого типа приведено для справки в приложении. Никаких других требований к структуре классов не налагается.

В нашем примере реализован очень простой учебный алгоритм, не требующий дополнительных комментариев. Исходный код класса `SharkOrderDetails` приведен на рис. 3.19.

По тем же правилам создаются классы `SharkDecline` и `SharkFinishOrder`, соответствующие приложениям `DoDecline` и `DoFinishOrder`. Их исходный код приведен на рис. 3.21 и рис. 3.22.

После успешного создания и компиляции классов их нужно поместить в jar-архив и разместить под управлением системы Shark. В Eclipse создание jar-архива выполняется достаточно просто, поэтому мы не будем сосредотачивать

на этом внимание. Созданный архив нужно поместить в один из следующих служебных каталогов системы Shark:

- ◆ %SHARK\_HOME%\lib;
- ◆ %SHARK\_HOME%\lib\contrib;
- ◆ %SHARK\_HOME%\lib\contrib\ext.

Для того чтобы классы из этого архива стали доступны в системе Shark, лучше всего перезапустить систему. И вообще, всякий раз, когда в служебный каталог помещается новая версия jar-архива, необходимо будет осуществлять перезапуск системы. Остановить систему Shark можно из администраторского приложения (команда «Connection/Shutdown engine» в основном меню), а запустить вновь —

```
package edu.hse.nnov.bi.shark;
import java.lang.Math;
import org.enhydra.shark.api.internal.toolagent.AppParameter;
public class SharkOrderDetails
{
    protected static String names [] = {"John Silver","Blue Martin","Keil Nelson"};
    protected static String accounts [] = {"111-B-222","333-C-786","693-V-476"};
    protected static final long MAX_INDEX = 2;
    protected static final long MAX_ORDER = 9898983;
    protected static final float MAX_AMOUNT = 10000000000.01F;
    public static void execute (AppParameter cust_name,
                               AppParameter cust_account,
                               AppParameter order_id,
                               AppParameter order_amount)
    {
        int index;
        try
        {
            System.out.println("SharkOrderDetails is running");
            index = (int) Math.round(Math.random() * SharkOrderDetails.MAX_INDEX);
            cust_name.the_value = new String(names[index]);
            index = (int) Math.round(Math.random() * SharkOrderDetails.MAX_INDEX);
            cust_account.the_value = new String(accounts[index]);
            order_id.the_value = new Long(1 +
                Math.round(Math.random() * SharkOrderDetails.MAX_ORDER));
        }
    }
}
```

**Рис. 3.19.** Исходный код для реализации класса SharkOrderDetails



```

order_amount.the_value = new Double(1.1+
    Math.random() * SharkOrderDetails.MAX_AMOUNT);
}
catch (Exception ex)
{
    System.out.println("SharkOrderDetails - Problems while executing procedure");
}
}
}
}

```

**Рис. 3.20.** Исходный код  
для реализации класса SharkOrderDetails (окончание)

программой %SHARK\_HOME%\bin\run. Естественно, что после перезапуска придется вновь проходить регистрацию в администраторском приложении.

После размещения jar-архива под управлением системы Shark потребуется поставить в соответствие приложения бизнес-процесса созданным классам. Это делается на закладке «Отображение программных систем на приложения бизнес-процессов» ('Application mapping') администраторского приложения.

При выборе этой закладки отображается список с созданными ранее соответствиями. Добавить новое соответ-

```

package edu.hse.nnov.bi.shark;
import org.enhydra.shark.api.internal.toolagent.AppParameter;
public class SharkDecline
{
    public static void execute (AppParameter cust_name,
                               AppParameter order_id)
    {
        try {
            System.out.println(" === DECLINE ORDER NOTIFICATION === ");
            System.out.println("Dear Customer:" + cust_name.the_value + ", your Order#" +
order_id.the_value.toString());
            System.out.println(" was Declined. Please communicate with our sales office.");
        } catch (Exception ex) {
            System.out.println("SharkDecline - Problems while executing procedure:" + ex);
        }
    }
}
}

```

**Рис. 3.21.** Исходный код  
для реализации класса SharkDecline

```

package edu.hse.nnov.bi.shark;
import org.enhydra.shark.api.internal.toolagent.AppParameter;
public class SharkFinishOrder
{
    public static void execute (AppParameter cust_name,
                               AppParameter order_id)
    {
        try
        {
            System.out.println(" ===== ORDER ACCEPTANCE NOTIFICATION
===== ");
            System.out.println(" Dear Customer:" + cust_name.the_value + ", your
Order#" + order_id.the_value.toString());
            System.out.println(" was Accepted.");
        }
        catch (Exception ex)
        {
            System.out.println("SharkFinishOrder - Problems while executing procedure." + ex);
        }
    }
}

```

**Рис. 3.22.** Исходный код для реализации класса SharkFinishOrder

ствие можно с помощью кнопки «Add». При этом появляется диалоговое окно «Create application definition to tool agent application mapping» (рис. 3.23).

Покажем на примере приложения GetOrderDetails, какие действия потребуется выполнить. В левой области диалога приведен список известных в бизнес-процессе приложений. Найдите и выделите приложение «GetOrder-Details»<sup>13</sup>. После этого в левой области экрана выберите из списка элемент «org.enhydra.shark.toolagent.JavaClassTool Agent». Тем самым вы определите тип программной системы, которая соответствует приложению «GetOrderDetails», — это java-класс. Про другие типы можно узнать в оригинальной документации. За выбором типа программной системы вам потребуется указать, какой конкретный класс будет вызываться для исполнения. Полное имя java-класса указывается в поле «Application name» (внизу справа). В нашем случае

<sup>13</sup> Если такое имя отсутствует, то это означает, что соответствующий пакет не был еще загружен в систему. Вернитесь к разделу, где обсуждается процедура загрузки пакетов.

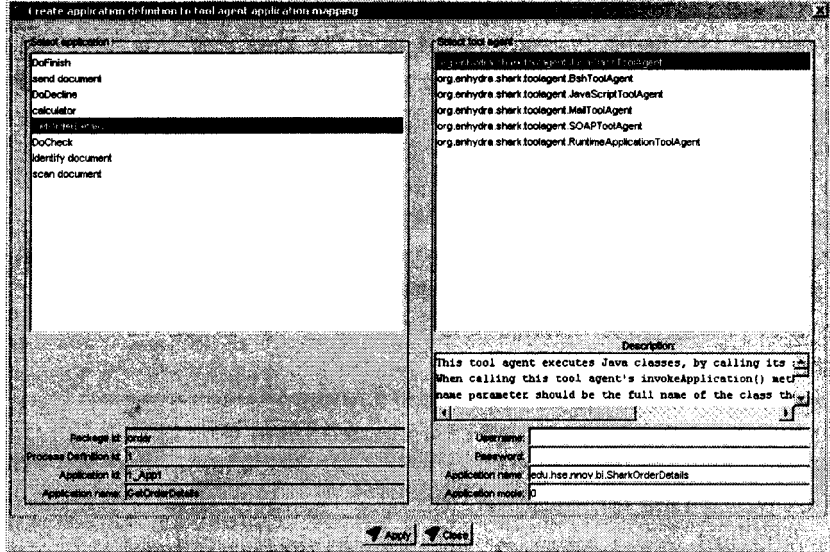


Рис. 3.23. Схема Диалог для определения соответствия между приложениями и программными системами

значение этого поля — `org.hse.nnov.bi.shark.SharkOrder Details`. Значение поля «Application mode» должно быть установлено в 0. Это приведет к тому, что распорядитель работ будет ожидать завершения работы метода класса.

Аналогично производится установка отображения для оставшихся приложений. В результате список соответствий на закладке должен выглядеть следующим образом (рис. 3.24):

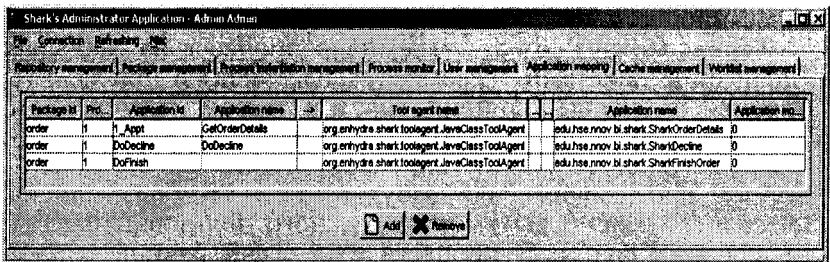


Рис. 3.24. Завершенный список соответствий

No transitions left to follow

2005-01-13 18:18:06,164: Process[key=1008\_order\_Check,mgrname=order#12#Check] - notifying requester of new state closed.completed

2005-01-13 18:18:06,164: Process[key=1007\_order\_1,mgrname=order#12#1] - Activity [Id=1017\_1007\_order\_1, ba=null, ActDefId=CheckAccount] is completed.

2005-01-13 18:18:06,184: JavaScriptEvaluator -> Javascript expression Order\_Stat us == "OK" is evaluated to true

2005-01-13 18:18:06,184: Process[key=1007\_order\_1,mgrname=order#12#1] - Activity [Id=1019\_1007\_order\_1, ba=null, ActDefId=Finish\_Order] is created

2005-01-13 18:18:16,759: Activity[Id=1019\_1007\_order\_1, ba=null, ActDefId=Finish\_Order] - Executing tool [Id=DoFinish]

===== ORDER ACCEPTANCE NOTIFICATION =====

**Dear Customer:Blue Martin, your Order#5500958 was Accepted.**

2005-01-13 18:18:16,769: Process[key=1007\_order\_1,mgrname=order#12#1] - Activity [Id=1019\_1007\_order\_1, ba=null, ActDefId=Finish\_Order] is completed.

**Рис. 3.25.** Результат выполнения приложения DoFinishOrder в том случае, когда оно соответствует классу edu.hse.nnov.bi.shark.SharkFinishOrder

Для полного знакомства с возможностями администраторского приложения нам осталось узнать, какие операции возможны на закладке «Управление пользователями системы Shark» («User management»). Вам предлагается изучить это самостоятельно и создать нового пользователя, которому будет поставлен в соответствие определенный участник бизнес-процесса.

Теперь можно создать экземпляр процесса на закладке «Управление пакетами» (Package Management) и проследить за процессом его выполнения. Как и в первом примере, вам потребуется выполнить серию действий на закладке «Управление назначенными для исполнения функциями» (Worklist management). Не забудьте при этом из списка «Select user» (рис. 3.11) выбрать нужного пользователя. Так как для всех функций процесса был задан ручной режим запуска, то остается обязательным подтверждение начала

исполнения каждой функции путем выделения поля «Accepted». Но теперь ручное изменение элементов данных при исполнении практически всех функций является необязательным. Например, при исполнении функции «receive order» вся инициализация данных процесса произойдет в результате работы приложения. Единственной функцией, исполняемой в ручном режиме, остается «Check\_credit», здесь вручную устанавливается нужное значение элемента «Credit\_Status». Отметим, что вывод всех текстовых сообщений в методах созданных вами java-классов будет осуществляться на консоль приложения Shark. Например, на рис. 3.25 выделен вывод из класса SharkFinishOrder.

### ***3.4. Контрольные задания к гл. 3***

1. Самостоятельно ознакомьтесь с возможностями клиентского приложения «Worklist Handler» (скрипт для запуска %SHARK\_HOME%\bin\runWH). Работа с этим приложением требует запущенного CORBA-сервера системы Shark.

2. Создайте и продемонстрируйте преподавателю работающий в системе Shark бизнес-процесс, используя один из XPDL-файлов, который вы разработали ранее в редакторе JaWE. В ходе выполнения этого задания создайте две различные версии одного и того же процесса. В одной версии все функции исполняются вручную, другая версия использует для исполнения приложения. Добавьте в описание процесса необходимые расширенные атрибуты, создайте и подключите к системе Shark разработанные вами java-классы.

3. Организуйтесь в группы по 2—3 человека и выберите для групповой работы один из процессов, созданных в задании № 2. В выбранной версии процесса функции должны выполняться вручную. Запустите на одной из машин систему Shark в режиме самостоятельного приложения, загрузите в нее описание процесса, создайте несколько пользователей (по числу участников группы) и назначьте им в произвольном порядке функции на исполнение. Про моделируйте групповой процесс выполнения бизнес-процесса, когда каждый участник группы работает за отдельным компьютером и использует свою версию приложения

«Worklist Handler» для исполнения назначенных функций. По результатам серии запусков подготовьте отчет по хронометражу выполнения процесса (для получения исходных данных можно использовать возможности администраторского приложения).

4. По документации и примерам изучите возможности и особенности использования скриптов в качестве приложений процесса. Измените настройки процесса из задания № 2 так, чтобы вместо java-классов для работы использовались скрипты. Необходимо соблюсти требование — скрипты должны находиться в отдельном `jar`-файле, а не в описании бизнес-процесса.

5. По документации и примерам изучите возможности и особенности использования особого типа `ToolAgent` — почтового клиента. Измените настройки процесса из задания № 4 так, чтобы исполнение одной из функций требовало отправки почты.

6. Установите последнюю доступную версию системы `Shark`. Изучите отличия и дополнения этой новой версии от описанной в гл. 3. Выполните в новой версии задания 1 и 5.

### ***3.5. Приложения к главе 3. Описание методов вызова внешних систем***

Консорциум `WfMC` предложил организовать вызов внешних программных и аппаратных компонент из системы `Workflow Management` на основе решения `Tool Agent` (мы предлагаем использовать русскоязычный термин «посредник по вызову внешних программных или аппаратных систем», или кратко — посредник). Каждый посредник служит для взаимодействия с конкретным типом внешней системы по определенному коммуникационному протоколу. Специфика взаимодействия с конкретным типом системы скрыта на уровне посредника, поэтому система `workflow management` передает запросы различным типам посредников через один и тот же стандартный интерфейс. Этот интерфейс разработан с учетом различных сценариев работы и позволяет осуществлять запуск и остановку приложений, контролировать их состояние во время работы, передавать данные процесса и предоставлять по запросам

приложений внутреннюю информацию. Полная спецификация интерфейса по вызову посредника включена в стандарт на WfMC-интерфейс № 3 (рис. 3.26).

Независимые разработчики программного и аппаратного обеспечения, желающие обеспечить взаимодействие их решений с системами workflow management, могут создать различные реализации посредников, предоставляю-

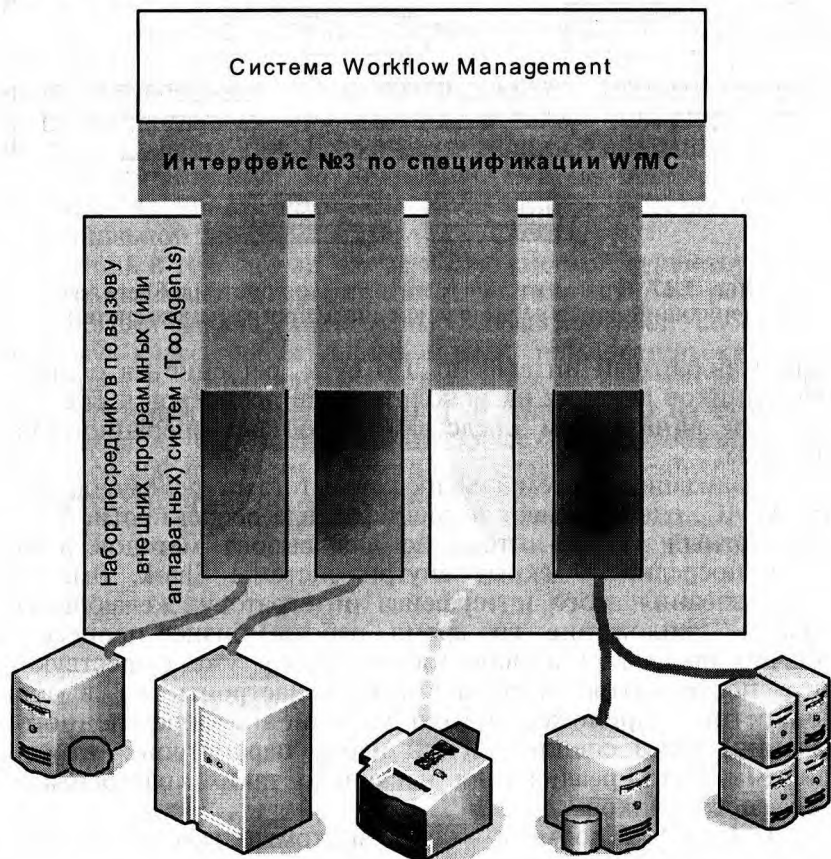


Рис. 3.26. Принципы вызова внешних программных систем по рекомендациям WfMC

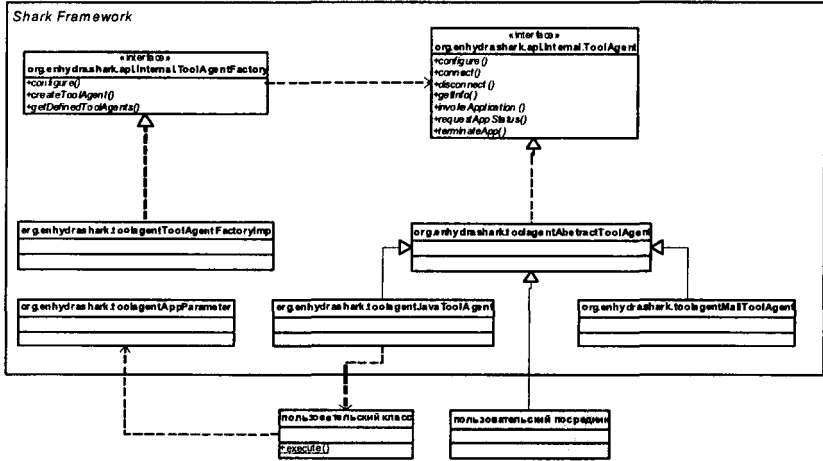


Рис. 3.27. Фрагмент диаграммы классов системы Shark, поясняющий способы вызова внешних программных систем

щих стандартный интерфейс. По сути, решение на основе посредников походит на использование драйверов к специфичным аппаратным средствам в составе операционной системы.

Реализация системы Shark соответствует рекомендациям WfMC, однако имеет и специфичные особенности. Так, стандартный WfMC-интерфейс для вызова методов внешних посредников скрыт внутри системы Shark. Вместо использования этого интерфейса интегратору, желающему вызвать приложение во время работы бизнес-процесса, обычно предлагается использовать набор уже существующих посредников, которые можно настроить на вызов конкретного приложения (или выполнение определенного задания) через специфический набор параметров. Авторы системы Shark реализовали несколько таких «настраиваемых» посредников:

1. `JavaClassToolAgent` — уже знакомый вам посредник для вызова произвольных java-классов;
2. `RuntimeApplicationToolAgent` — посредник для вызова внешних приложений;



3. JavaScriptToolAgent — посредник для выполнения Java-скриптов;

4. BshToolAgent — посредник для выполнения скриптов на языке Java;

5. SOAPToolAgent — посредник для вызова удаленных объектов по протоколу SOAP;

6. MailToolAgent — посредник для получения и отправки почты.

Каждый такой посредник по разному интерпретирует значения стандартных параметров (например, параметры «Application Name», «Application Mode»), используемых в администраторском приложении, а также применяет различные правила преобразования входных-выходных параметров приложения. За подробностями по использованию каждого из «настраиваемых» посредников обращайтесь к оригинальной документации.

Лишь в том случае, когда возможностей существующих «настраиваемых» посредников недостаточно для решения конкретной задачи, необходимо создавать свой собственный java-класс, реализующий новый пользовательский посредник. В этом случае разработчик должен следовать определенным рекомендациям и ограничениям на структуру класса. В частности, новый класс должен обязательно являться наследником системного класса `org.enhydra.shark.toolagent.AbstractToolAgent` (рис. 3.27).

---

---

## Глава 4 \_\_\_\_\_

### Архитектура и технология использования современных ERP-систем на примере системы OFBIZ

#### 4.1. Введение

Изучив раздел по автоматизированному управлению процессами (Work-flow Management Software), вы познакомились с назначением и основными характеристиками этого важного класса программного обеспечения. В его состав входят:

- ◆ специализированные графические редакторы, в наглядной форме представляющие всю структуру и характеристики бизнес-процесса;

- ◆ особые языки декларативного типа (например, язык XPDL), позволяющие в формальном и переносимом с системы на систему виде выражать порядок исполнения отдельных функций, правила переходов между ними в зависимости от значений определенных элементов данных, установленные связи работников предприятия и программных систем с выполняемыми функциями;

- ◆ распорядители работ (Workflow Engines), использующие такие описания процессов для непосредственного исполнения функций процесса, вызывая различные программные системы, уведомляя пользователей о назначении им новых заданий, собирая статистику о ходе выполнения процесса.

Комплексное использование средств автоматизации бизнес-процессов позволяет сделать работу предприятий более прозрачной для руководителей и рядовых сотрудников,

повышая их заинтересованность в выполнении назначенных функций. Появляется также возможность в ходе имитационного моделирования или реального выполнения процессов объективно оценивать затраты на исполнение каждой функции, ее влияние на эффективность деятельности предприятия, а расширяемость и способность к динамичному переконфигурированию, присущие программным средствам класса Workflow Management, позволяют просто перестраивать порядок и средства выполнения процесса с целью его постоянного улучшения и соответствия заданным требованиям производительности.

Но при всем этом средства автоматизации бизнес-процессов сами по себе еще недостаточны для построения настоящей корпоративной информационной системы (КИС). Есть ряд возможностей, которые либо полностью отсутствуют в этих средствах, либо недостаточно развиты. К таким возможностям, оказывающим существенное влияние на эффективное применение КИС, нужно отнести:

### *1. Возможность расширения информационной модели.*

Обычно большая часть программного обеспечения информационной системы создается единожды. В то же время она должна удовлетворять требованиям многочисленных групп пользователей, нуждам предприятий различных сфер деятельности. Прежде всего, отличаются требования к структуре и содержанию информации, которая нужна конкретной группе пользователей. Наверное, понятно, что немыслимо заранее предусмотреть все возможные варианты использования информационной системы и включить в ее состав все потенциально полезные элементы данных (структуры данных, таблицы, шаблоны документов и т.п.). Поэтому в составе КИС должны содержаться средства по расширению или изменению исходной информационной модели. Эти средства должны позволять описание в декларативной форме новых типов данных, которые становятся нужны для выполнения вновь появляющихся задач, а также стабильный и единообразный интерфейс для создания, удаления и модификации любых описанных элементов данных во всех компонентах эксплуатируемой системы. Стоит добавить, что в отличие от средств автоматизации бизнес-процессов, где уже существует возможность определения простейших

элементов данных, в КИС требуется выражать более сложные структуры данных, определять взаимосвязи между различными типами данных и правила по долговременному хранению информации в различных базах данных.

2. *Единая технология для реализации прикладных алгоритмов по обработке данных и интеграции.* Несмотря на то, что разработчики КИС обычно стремятся интегрировать в систему большую часть существующих на предприятии программных систем и алгоритмов, им приходится создавать большое количество программных компонентов заново. Чтобы затраты времени на их создание были минимальны, а программная реализация разных алгоритмов была максимально унифицирована (это облегчает поддержку и повышает коэффициент повторной исполняемости кода), в системе должна иметься хорошо определенная технология создания программного обеспечения. Такая технология обычно включает в себя служебные библиотеки, упрощающие доступ к внешним данным и реализацию сложных алгоритмов, особые языки программирования высокого уровня для специалистов в предметной области, автоматизированные средства построения пользовательского интерфейса, инструментарий для ведения каталогов созданных модулей, удобные программы для настройки.

3. *Развитые средства графического интерфейса пользователя с единым корпоративным стилем.* Использование КИС предполагает, что к одним и тем же данным будет обращаться большое количество пользователей на различных уровнях структуры предприятия для решения многочисленных задач. Чтобы обеспечить согласованность процедур обработки данных, сократить время на переобучение персонала, повысить чувство сопричастности пользователя к деятельности предприятия нужно, чтобы все пользователи системы использовали в своей работе с КИС дружественные графические интерфейсы, специально разработанные для наиболее эффективного выполнения конкретной задачи, но при этом обладающие единством стилового решения. Последнее означает, что во всех графических программах КИС должны использоваться единые правила размещения основных рабочих областей (меню, строка статуса, стандартные кнопки, система помощи и т.п.), согласованные цве-

товые решения для индикаторов, графических изображений, одинаковые наименования для одного и того же элемента данных или операции, одинаковые диалоги для ввода данных и т.д. и т.п. Примем во внимание, что одно и то же программное обеспечение может использоваться для реализации КИС различных предприятий. Поэтому возникает то же требование расширяемости, что и в п. 1: КИС должна обладать возможностью декларативного определения произвольного нового интерфейса пользователя, согласованного с ранее разработанным корпоративным дизайном и общими правилами размещения рабочих элементов, с последующей динамичной привязкой элементов этого интерфейса к выполняемым операциям или данным. Кроме того, развитые КИС поддерживают каталоги графических интерфейсов и их составных частей, управляют доступом к интерфейсам в зависимости от категории пользователя, обеспечивают управление содержанием корпоративных web-сайтов и т.п.

*4. Наличие в системе элементов данных и прикладных алгоритмов для выполнения стандартных процедур.* Повседневные задачи на предприятии решаются на основе законов, стандартных регламентов, нормативов и других общегосударственных (или отраслевых) руководящих документов. Они однозначно определяют большую часть требований на используемые данные, содержание бизнес-процессов и алгоритмов обработки данных, оставляя лишь небольшую свободу и неопределенность для учета специфики отдельного предприятия. В таком случае целесообразно заранее включить в состав программного обеспечения КИС определения стандартных элементов данных и реализацию общих для всех потенциальных пользователей процессов и алгоритмов, сгруппировав определения, процессы и алгоритмы по функциональному признаку в отдельные компоненты. Тогда даже без настройки под конкретное предприятие КИС будет обладать способностью поддержки большинства основных задач (а пользователи при покупке системы смогут выбрать минимально достаточный для них набор компонентов). Чаще всего такими общими компонентами, с самого начала входящими в КИС, являются: бухгалтерский учет, отдел кадров, складской учет, обработка заказов,

инвентаризация, доставка товаров. Кроме того, КИС может иметь компоненты, которые реализуют стандартные бизнес-процессы и алгоритмы какой-то определенной отрасли (тяжелая промышленность, нефтепереработка, муниципальное хозяйство, транспорт и т.п.).

Даже краткое рассмотрение наиболее важных возможностей КИС служит наглядной иллюстрацией их повышенной сложности и множества решаемых задач. Без преувеличения можно сказать, что в настоящее время КИС на своем высшем уровне развития (ERP- и CSRP-системы) являются одним из важнейших практических результатов в информатике и программной инженерии.

В последующих главах и практических заданиях нам предстоит познакомиться с основными составляющими архитектуры современных ERP-систем, принципами их практического использования, настройки под конкретные задачи и расширения функциональности.

## ***4.2. Общая информация о системе OFBIZ***

OFBIZ (от Open for Business) является набором библиотек и приложений на языке Java, разрабатываемых и свободно распространяемых независимой международной группой разработчиков. Родившись как исследовательский проект, система OFBIZ, развиваясь, достигла функционального уровня коммерческой ERP-системы. Центр обучения и управления разработкой находится в США (домашняя страница в Интернете — <http://www.ofbiz.org>), предлагая свободный доступ к исходному коду, а обучение, документацию по системе и доработку по требованиям конкретного заказчика — на платной основе. Такая политика привлекает к работе по тестированию, исправлению ошибок и расширению функциональности большое количество независимых разработчиков и компаний-интеграторов со всего мира, в число которых входит и группа из Нижегородского филиала ГУ-ВШЭ. В своем составе OFBIZ использует большое количество бесплатных библиотек сторонних производителей, являясь удачным примером того, как разумная интеграция небольших по функциональности подсистем может приводить к появлению в системе новых качеств.

Практически все наиболее известные базовые информационные технологии<sup>14</sup>, реализованные в библиотеках сторонних производителей, в системе OFBIZ образуют завершённый ансамбль, предлагающий конечному пользователю следующие функциональные компоненты<sup>15</sup>.

1. Электронная коммерция (e-commerce):

- a) реализация Интернет-витрин и Интернет-магазинов;
- b) поиск товаров по ключевым словам;
- c) категоризация товаров;
- d) управление корзиной покупателя в Интернет-магазине (shopping cart);
- e) настраиваемый механизм скидок на товары;
- f) ведение профиля каждого покупателя;
- g) гибкое ценообразование (в зависимости от профиля покупателя, скидок и др. информации).

2. Ведение информации по клиентам и работникам (Party Manager):

- a) индивидуальные пользователи и группы пользователей;
- b) поиск по ключевым словам;
- c) личная и деловая информация;
- d) безопасность (пароли, ключевые слова);
- e) механизмы взаимодействия (почта, адреса и т.п.);
- f) механизмы оплаты (скидки, пакеты товаров по специальным ценам);
- g) назначение ролей для работы с системой OFBIZ.

3. Управление маркетинговой деятельностью (Marketing Manager):

- a) проведение маркетинговых компаний через веб-сайт;
- b) статистика по спросу на отдельные товары и их различные сочетания.

4. Управление каталогами продукции (Catalog Manager):

- a) настраиваемая категоризация товаров;
- b) электронные каталоги (изображения товаров, описания, размещение, доступность для покупателя и т.п.);
- c) управление свойствами товаров и конфигурацией товаров;

---

<sup>14</sup> Например: Sun J2EE, W3C XML, HTML, SOAP, WfMC XPD, OMG GL, и др.

<sup>15</sup> Информация на январь 2007 г.

d) управление ценовой политикой и скидками.

5. Управление недвижимостью предприятия (Facility Manager):

a) складской учет;

b) инвентаризация помещений;

c) ведение расписаний занятости помещений.

6. Управление доставкой товаров заказчикам (Shipment Manager):

a) автоматическое создание договора на доставку на основании заказа;

b) ведение расписаний доставок товаров покупателям;

c) взаимодействие с известными службами доставки (например, UPS);

d) поиск доставки, определение ее текущего статуса.

7. Обработка заказов (Order Manager):

a) поиск заказов, получение текущего статуса;

b) ввод новых заказов;

c) обработка возвратов товара;

d) управление скидками после покупки (refunds).

8. Бухгалтерский учет (Accounting Manager):

a) ведение списка счетов;

b) ведение счетов-фактур;

c) автоматизация бухгалтерских проводок.

9. Автоматизация бизнес-процессов (Work Effort and Workflow Management):

a) управление списком задач для выполнения определенным пользователем;

b) определение календарных событий;

c) определение и управление функциями бизнес-процесса по стандарту WfMC (поддерживается язык XPDL);

d) поддержка службы помощи (Help Desk) и автоматизированная обработка запросов к этой службе (на информацию о товаре, на исправление ошибки в программе, на ремонт компьютера и т.п.).

10. Управление публикацией данных на корпоративном web-сайте (Content Manager):

a) поддержка виртуальных WEB-сайтов на одном сервере;

b) управление динамическими обзорами продукции на сайте;



- с) облегчение доступа к информации из баз данных;
- d) голосование через WEB-сайт;
- е) поиск ресурсов на сайте;
- f) поддержка единого корпоративного стиля;
- g) высокоуровневые средства редактирования;
- h) назначение ответственных и контроль за подготовкой материалов сайта.

11. Управление производством (Manufacturing Manager):

- a) разработка списка материалов на производство;
- b) размещение заказов на производство;
- с) контроль поступления материалов и продукции.

12. Управление точками продажи (Point Of Sale Manager):

- a) связь с электронными терминалами;
- b) автоматическая загрузка каталогов и ценовой информации.

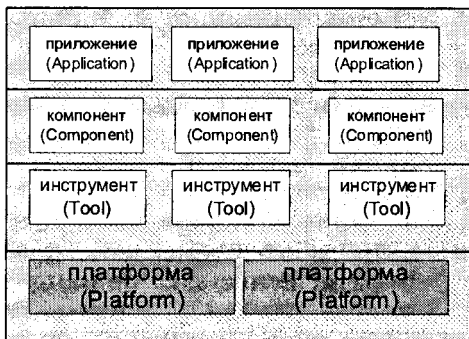
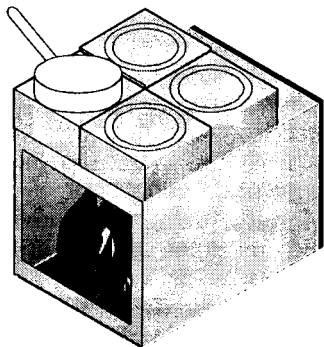
13. Авторизация и аутентификация пользователей (Security Manager):

- a) одноразовая регистрация;
- b) разграничение прав доступа на основе ролей;
- с) автоматическое переключение режимов безопасности при доступе к разным подсистемам.

Все функциональные компоненты используют похожие принципы организации, средства описания используемых данных и технологии разработки алгоритмов по их обработке. Интерфейс пользователя во всех функциональных компонентах обладает согласованным дизайном, разграничением по правам доступа и основан на применении WEB-технологий, давая возможность работы через стандартный Web-браузер. Кроме конечного пользователя к системе OFBIZ могут обращаться другие программные системы через стандартные интерфейсы технологии J2EE.

### ***4.3. Краткий обзор архитектуры OFBIZ***

OFBIZ является примером системы, построенной по технологии «клиент-сервер» и сочетает в себе каркасы и средства разработки новых компонентов как на стороне «Back-End» — (сервер), так и на стороне «Front-End» (клиент). Сервер, созданный на языке Java и использующий серверные технологии J2EE, предоставляет внутренним и



**Рис. 4.1.** Метафора плиты и ее воплощение в делении архитектуры на слои

внешним клиентам набор различных функций, оформленных в виде служб (сервисов).

Внутренняя структура сервера OFBIZ чрезвычайно модулярная, что позволяет достаточно просто ее настраивать и расширять для выполнения конкретных задач путем редактирования XML-файлов. Деление сервера на модули происходит в соответствии с так называемой архитектурой плиты (Stovepipe Architecture, рис. 4.1).

Метафора плиты подчеркивает, что единая платформа предоставляет базовые услуги (огонь в плите предоставляет тепло для приготовления всех блюд), базовые услуги используются в разной степени (на плите несколько отделений с разным размером и температурой) для решения всевозможных пользовательских задач (на каждом отделении плиты может готовиться особая пища).

В архитектуре OFBIZ определено четыре горизонтальных логических слоя, а реализацию функциональности каждого слоя осуществляют несколько слабо связанных элементов. Элементы каждого слоя играют вполне определенную роль:

1) на уровне платформы расположены элементы, реализующие базовые информационные технологии (работа с файлами, доступ к базам данных, взаимодействие по сети);

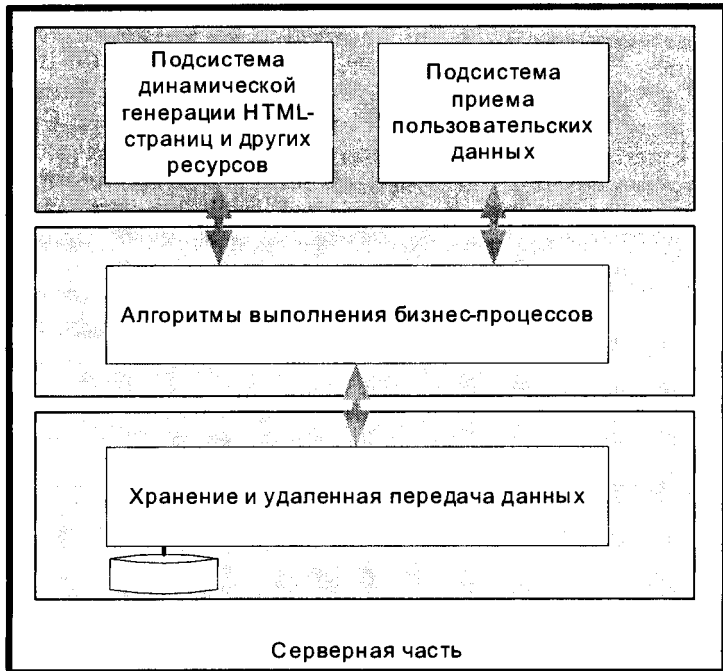


Рис. 4.2. Общепринятое деление на слои серверной части

2) на уровне общих инструментальных средств решаются задачи общего управления (диспетчеризация вызовов, совместное использование данных, защита информации и т.п.);

3) на уровне общих прикладных компонентов находятся алгоритмы и определения данных для решения определенных прикладных задач, оформленные как службы;

4) уровень приложений реализует графический специализированный интерфейс для удобного доступа пользователей к определенным функциям системы.

Используя традиционное деление серверных программных систем на три уровня, можно считать, что уровни 1 и 2 вместе образуют уровень данных, уровень 3 образует уровень бизнес-логики, а уровень 4 — уровень представления (рис. 4.2).

На рис. 4.3 и далее в диаграммах светло-серый цвет обозначает изменяемые при настройке элементы, а темно-серый цвет — стандартные элементы системы OFBIZ и элементы базовых информационных технологий. Наиболее значимые и оригинальные составляющие системы OFBIZ сосредоточены на уровне общих инструментальных средств. А среди всех элементов этого уровня необходимо, прежде всего, обратить внимание на Service Engine и Entity Engine.

Предлагается использовать для Service Engine русскоязычное название «Центр управления службами» (ЦУС), а для Entity Engine — «Центр управления элементами данных» (ЦУЭД). ЦУС обеспечивает каркас для реализации

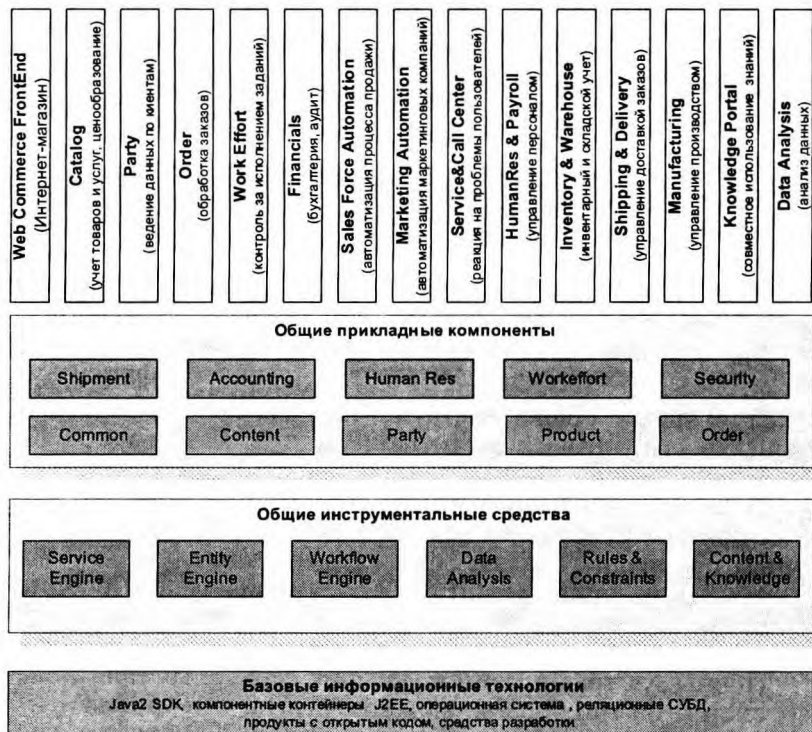


Рис. 4.3. Декомпозиция уровней системы OFBIZ

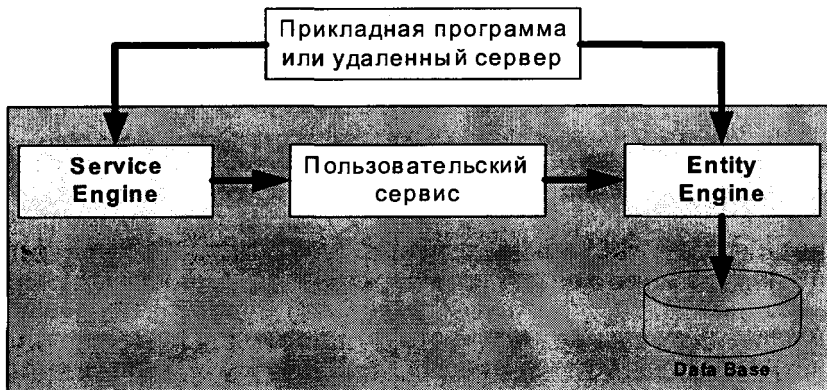


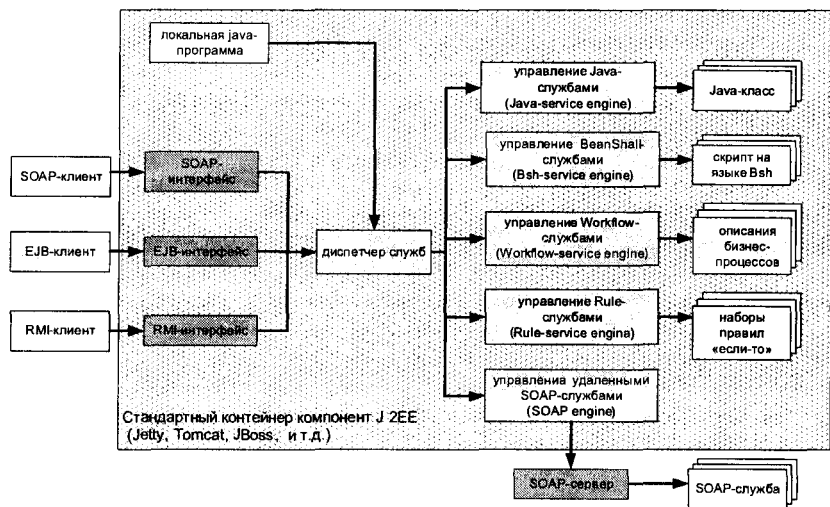
Рис. 4.4. Концепция разделения полномочий в системе OFBIZ

логики, а ЦУЭД предоставляет единые для всей системы средства по определению и работе с данными (рис. 4.4).

#### 4.3.1. *Service Engin* — центр управления службами (ЦУС)

Концепция службы (Service) в настоящее время является одной из важнейших при построении распределенных корпоративных информационных систем. Достаточно посмотреть в Интернете на количество публикаций и примеров реализаций так называемых Web-служб (Web-Services). Можно даже сказать, что был разработан и стал активно использоваться отдельный шаблон проектирования. Сущность этого шаблона — понятие «служба» — отдельный несложный процесс, который предназначен для выполнения одного или нескольких специфических действий. Правила взаимодействия службы с многочисленными независимыми клиентами полностью задаются определением службы (Service Defintion)<sup>16</sup>. Обычно определение службы — это набор выражений на каком-нибудь формальном языке декларативного типа, в котором описаны имена функций, имена и типы всех входных и выходных параметров и т.п.

<sup>16</sup> При этом детали практической реализации службы полностью скрыты от клиентов, поэтому разработчики могут использовать различные языки программирования и технологии, наиболее подходящие для решения текущей задачи.



**Рис. 4.5.** Назначение центра управления службами в системе OFBIZ

Определение службы открыто для всех потенциальных пользователей (например, оно может быть загружено с Web-сайта). Для взаимодействия клиента и службы используется один из стандартных сетевых протоколов, а значения фактических параметров, переданных клиентом в службу, проверяются на соответствие типа перед реальным вызовом функции. Точно так же результат выполнения функции проходит проверку на непротиворечивость и соответствие объявленному типу перед передачей клиенту.

Именно с использованием концепции службы в системе OFBIZ выполняется реализация всех алгоритмов бизнес-логики. Любой прикладной алгоритм, любое специфическое действие оформляются в виде отдельной службы. Описание службы в системе OFBIZ строится по определенным правилам на языке XML, а для реализации методов службы может быть выбран один из нескольких вариантов:

- ◆ статический публичный метод Java-класса;
- ◆ скрипт на языке интерпретируемого типа (например, Java-скрипт);

- ◆ скрипт на проблемно-ориентированном языке в формате XML (так называемый Mini-Language);
- ◆ описание бизнес-процесса;
- ◆ набор условных продукций ЕСЛИ-ТО;
- ◆ вызов любой внешней службы по стандартному протоколу SOAP.

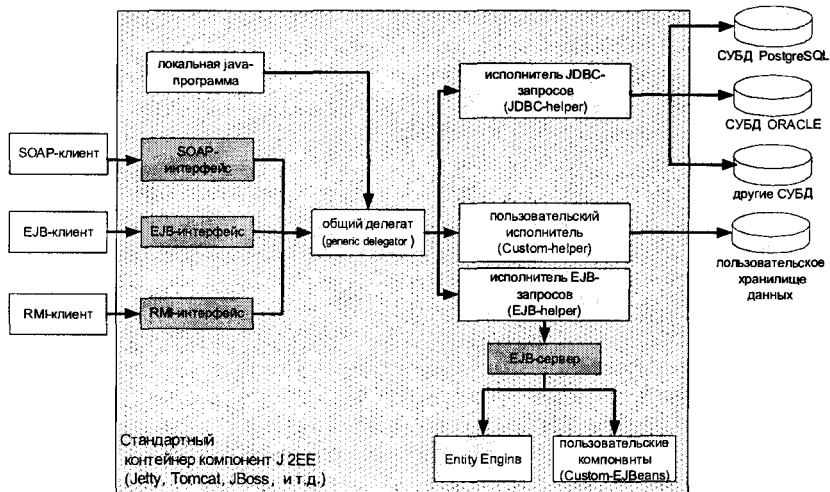
В ЦУС проведена реализация единого внутреннего интерфейса по вызову функций служб и передаче-приему параметров. Таким образом, для вызова службы внутри системы достаточно знать символическое имя службы и требуемые для работы параметры. Кроме того, ЦУС позволяет задавать правила автоматического вызова службы при наступлении определенных условий внутри системы (так называемые правила «событие-условие-действие», в оригинале — Event-Condition-Action rules) или назначать автоматический запуск службы на определенное время. ЦУС также позволяет обращаться к службам системы OFBIZ различным внешним клиентам по стандартным протоколам SOAP, RMI и EJB (рис. 4.5).

#### 4.3.2. *Entity Engine — центр управления элементами данных (ЦУЭД)*

В определении ЦУЭД важнейшую роль играет понятие «элемент данных» (или сущность, от Entity). Под элементом данных понимают любое мыслимое понятие или объект реального мира, с которым может работать информационная система, полностью определяемый с помощью имени и набора определенных именованных свойств. Элемент данных может быть логически связанным с другими элементами данных, образуя сложную взаимосвязанную структуру<sup>17</sup>. Для выполнения всех операций по созданию, модификации, поиску элементов данных и навигации по связям в ЦУЭД используется собственный интерфейс, который не зависит от применяемой технологии баз данных. Таким образом, например, при работе с данными в системе OFBIZ не требуется знания языка SQL (рис. 4.6).

---

<sup>17</sup> Для тех, кто знаком с терминологией баз данных, уточним, что таким образом в системе OFBIZ реализуется сильно нормализованная реляционная модель данных.



**Рис. 4.6.** Назначение центра управления элементами данных в системе OFBIZ

Есть еще одно отличительное свойство ЦУЭД — в ее состав включена богатая модель данных, учитывающая специфику и потребности в информации различных ERP- и CRM-систем. Структура этой модели построена с учетом рекомендаций, изложенных в известной книге «The Data Model ResourceBook» [8], а также спецификаций OMG, и включает в себя все важнейшие элементы данных и взаимосвязи между ними, требующиеся при выполнении большинства типичных бизнес-процессов. По этой причине заметно повышается коэффициент повторной используемости кода и данных, сокращается избыточность данных и появляется реальная возможность экспорта и импорта данных из работающих баз данных различных предприятий. Нужно отметить, что модель данных описывается в системе OFBIZ в декларативной форме с помощью XML, поэтому независимые разработчики и администраторы могут самостоятельно изменять и расширять набор элементов данных по потребности. При этом от них практически не требуется знаний специфики работы с различными базами данных.



В настоящий момент в стандартной модели данных системы OFBIZ определено порядка 460 различных взаимосвязанных элементов данных, сгруппированных по следующим пакетам:

◆ **Common** — общесистемные данные (географические понятия, единицы измерения, стандартные перечисления, денежные единицы и т.п.);

◆ **Content** — учет различных ресурсов, связанных с подготовкой и отображением информации на web-сайте (HTML-страницы, шаблоны, графические картинки, документы и т.п.);

◆ **Party** — данные о клиентах, работниках, организационной структуре предприятия, разделении ответственности при выполнении функции и т.п.;

◆ **Product** — различная информация о продуктах, необходимая при покупке, продаже, производстве, складировании, классификации. Продуктами могут являться материальные предметы, услуги, электронная информация и пр.;

◆ **Facility** — информация о структуре недвижимости на предприятии. В эту категорию относят сведения о структуре зданий (помещения различного типа, их характеристики, коммуникации в здании и т.п.), организации складов, методах складирования различных продуктов;

◆ **Order** — информация, необходимая для оперативной и качественной обработки заказов, поступающих от клиентов;

◆ **Shipment** — данные, определяющие различные процедуры доставки купленных товаров и позволяющие осуществлять постоянный контроль за процессом доставки от склада до покупателя;

◆ **Accounting** — данные для организации компьютеризованного бухгалтерского учета;

◆ **Marketing** — информация о проводимых маркетинговых компаниях, списки рассылки рекламной информации;

◆ **Human Resources** — данные для учета сведений о работниках (должности, служебные обязанности, квалификация, условия оплаты труда и т.п.);

◆ **Work Effort (Workflow)** — данные, необходимые для организации автоматизированной службы помощи в разре-

шении проблем (Service Desk), организации Workflow Management;

◆ Security — данные для автоматизации контроля доступа к функциям системы. Сюда относится информация о регистрационных именах пользователей, их паролях, протоколы работы пользователей (log), списки полномочий.

### 4.3.3. Подсистема пользовательского Web-интерфейса

В системе OFBIZ технология создания пользовательского интерфейса основана на различных методах генерации динамических HTML-страниц (сервлеты, JSP-страницы и т.п.). Поэтому обязательным условием для использования интерфейса является работа OFBIZ в составе какого-то контейнера J2EE<sup>18</sup> и наличие у пользователя стандартного Web-браузера.

В механизмах генерации страниц и доступа из них к данным и службам чрезвычайно важную роль играют шаблоны проектирования «Модель-вид-контроллер», «Делегат» и «Декоратор». Они позволяют изолировать задачи разработки внешнего вида интерфейса от проектирования и реализации внутренних алгоритмов обработки данных, а также обеспечить высокую повторную используемость стандартных элементов стилового оформления страниц (можно совместно использовать стандартные меню, заголовки и «подвалы» страниц, диалоги и т.д.).

Рассмотрим порядок обработки пользовательского запроса на доступ к определенной HTML-странице в системе OFBIZ (рис. 4.7). Как вы знаете из материалов по Java-апплетам, каждый ресурс в Интернете (например, HTML-страница) имеет уникальный идентификатор — URI. Именно его вводит пользователь в поле адреса стандартного Web-браузера для начала работы. Обычно URI состоит из протокола, имени сервера, сетевого порта, каталога, обозначающего определенный модуль системы OFBIZ, и имени требуемого интерфейса. Последний элемент URI может отсутствовать (например, <http://localhost:8080/ecommerce/>).

---

<sup>18</sup> Система OFBIZ поставляется вместе с полнофункциональным J2EE-контейнером Tomcat, поэтому отдельно искать и устанавливать его не нужно.



1. запрос ресурса по URI

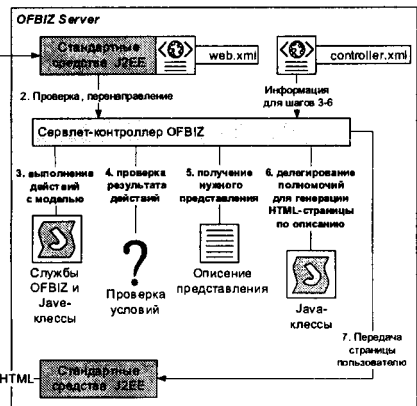
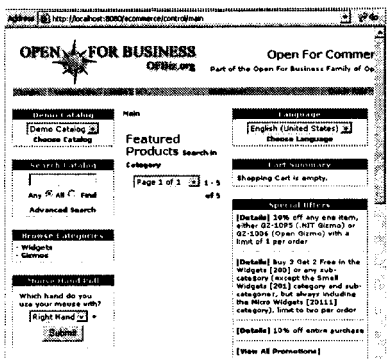


Рис. 4.7. Принципы функционирования подсистемы пользовательского интерфейса в системе OFBIZ

Поступивший на сервер URI попадает в модуль безопасности, где определяется, имеет ли пользователь право на доступ к запрошенному ресурсу. Если ресурс общедоступный и не требуется предварительно вводить пароль и секретное слово (либо пользователь уже зарегистрирован)<sup>19</sup>, то сервер определяет, каким образом запрошенный ресурс будет возвращен пользователю.

Статические ресурсы, которые не могут изменяться (например, графические изображения, статичные HTML-страницы или заранее созданные pdf-файлы), сразу же возвращаются сервером пользователю. Правила определения таких ресурсов разработчик или администратор могут самостоятельно изменить или дополнить в ходе настройки системы путем модификации XML-файла конфигурации web.xml.

Для доступа к динамичным ресурсам производится перенаправление исходного URI с целью включения в работу системы OFBIZ (рис. 4.7).

<sup>19</sup> Если ресурс защищен паролем, то автоматически выполняется перенаправление (redirect) на страницу для ввода регистрационной информации.

Обычно URI изменяется путем явного добавления подстроки control и имени требуемого интерфейса (например, <http://localhost:8080/ecommerce/control/main> — здесь требуется интерфейс с именем main). В соответствии с внутренними установками сервера модифицированный URI всегда обрабатывается одним и тем же сервлетом — сервлетом-контроллером интерфейса. Напомним, что сервлет — это служебный java-класс, реализующий особый интерфейс и работающий в составе J2EE-контейнера. Контейнер вызывает нужный метод сервлета-контроллера, передавая в качестве аргументов URI, настройки пользователя и другие параметры, и ожидает в качестве результата готовый текст HTML-страницы для передачи на Web-браузер пользователя.

Весь алгоритм генерации HTML-страницы в сервлет-контроллере определяется конфигурацией, хранящейся в XML-файлах системы OFBIZ. Таким образом, администратору или программисту, расширяющему функциональность системы, изменять нужно лишь структуру и содержимое XML-файлов, не касаясь Java-кода сервлета-контроллера.

Наиболее важный файл — это WEB-INF/controller.xml. В этом файле каждому возможному имени интерфейса ставится в соответствие набор действий и параметров, по которым полностью можно построить законченную HTML-страницу. В системе OFBIZ оно имеет название «requestmap» и ему соответствует отдельный XML-тег. Рассмотрим, например, содержимое тега <requestmap> для интерфейса «checkLogin». Этот интерфейс позволяет пользователю ввести регистрационное имя и пароль (рис. 4.8).

```
<request-map uri="checkLogin" edit="false">
  <description>Verify a user is logged in.</description>
  <security https="true" auth="false"/>
  <event type="java" path="org.ofbiz.securityext.login.LoginEvents"
  invoke="checkLogin"/>
  <response name="success" type="view" value="main"/>
  <response name="error" type="view" value="login"/>
</request-map>
```

Рис. 4.8. Фрагмент файла controller.xml с определением правил обработки запроса с именем «checkLogin»

Интерес представляет внутренний тег `<security>`. Его содержимое в нашем примере определяет, что в дальнейшем передача данных от клиента к серверу будет осуществляться по защищенному сетевому протоколу (`http-Secured`), а для доступа к данному интерфейсу не требуется предварительной регистрации. Далее располагается внутренний тег `<event>`, определяющий необходимость вызова метода `«checkLogin»` в классе `«org.ofbiz.securityext.login.Login Events»`. В этом методе происходит вся реальная работа по проверке введенных пользователем данных регистрации. Таким образом, в соответствии с шаблоном «модель-вид-контроллер» класс `«org.ofbiz.securityext.login.Login Events»` предоставляет интерфейс к модели. В зависимости от результатов проверки метод `«checkLogin»` может вернуть текстовую строку `«success»` или `«error»`. Правила реакции на каждый из возможных вариантов ответа определены во внутреннем теге `<response>`.

```
<view-map name="main" type="screen"
page="component://ecommerce/widget/CommonScreens.xml#main"/>
```

**Рис. 4.9.** Фрагмент файла `controller.xml` с определением типа и пути доступа к описанию представления с именем «main»

Тег `<response>` связывает определенный вариант ответа с символическим именем одного из разработанных представлений (`view`). Представление чаще всего определяет содержимое одной HTML-страницы, возвращаемой контроллером. В нашем примере при удачной регистрации пользователю возвращается представление с именем «main», а при ошибке — представление с именем «login». Связь символического имени представления с детальным определением его структуры также находится в файле `controller.xml`. Для задания связи используется тег `<view-map>` (рис. 4.9).

Из рис. 4.9 видно, что атрибут «page» определяет местоположение определения структуры представления. В нашем случае определение представления «main» находится в отдельном XML-файле, к которому можно получить доступ по URI `«component://ecommerce/widget/CommonScreens.xml»`.

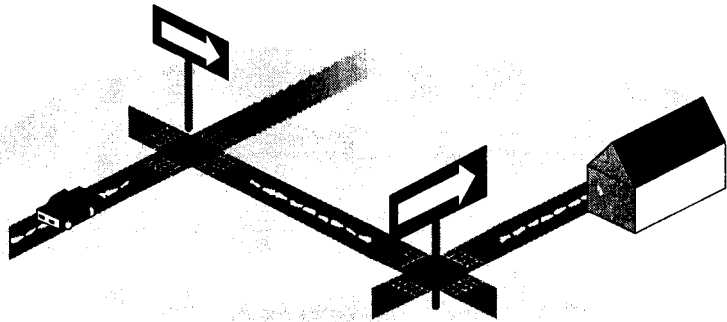
```
<handler name="jsp" type="view"
class="org.ofbiz.content.webapp.view.JspViewHandler"/>
<handler name="ftl" type="view"
class="org.ofbiz.content.webapp.ftl.FreeMarkerViewHandler"/>
<handler name="http" type="view"
class="org.ofbiz.content.webapp.view.HttpViewHandler"/>
<handler name="screen" type="view"
class="org.ofbiz.content.webapp.view.ScreenWidgetViewHandler"/>
```

**Рис. 4.10.** Фрагмент файла `controller.xml` с определением привязки типов представлений к обработчику

Все описание этого представления выражено на специальном языке «OFBIZ-Widgets» (на это указывает значение атрибута «type» — «screen»). В системе OFBIZ существует служебный класс-обработчик, который умеет по описанию на языке «OFBIZ-Widgets» создавать правильные HTML-страницы. Сервлет-контроллер делегирует полномочия по генерации страницы этому классу-обработчику, передавая URI определения, и получает уже готовый результат. Использование языка «OFBIZ-Widgets» — не единственная возможность определения структуры представления. Представления могут определяться различным образом: в виде JSP-страниц, HTML-страниц с удаленных серверов и т.д. Пользователь может даже придумать собственный язык описания структуры и создать соответствующий класс-обработчик. Поэтому атрибут «type» тега `<view-map>` может содержать различные символические константы, помогающие контроллеру определить, какому классу следует делегировать функции по генерации страницы. Связь каждой разрешенной к использованию в атрибуте «type» символической константы задается отдельным тегом `<handler>` в файле `controller.xml` (рис. 4.10).

Из рис. 4.10 видно, что описания представлений на языке «OFBIZ-Widgets» (тип — «screen») обрабатываются в классе `org.ofbiz.content.webapp.view.ScreenWidgetViewHandler`, а HTML-страницы с удаленных серверов (тип «http») — в `org.ofbiz.content.webapp.view.HttpViewHandler`.

Таким образом, в файле `controller.xml` действительно в декларативной и понятной форме определены все шаги по



**Рис. 4.11.** Подобно указателям на перекрестках опосредованные вызовы методов через интерфейсы позволяют динамично конфигурировать путь обработки запросов и применение различных алгоритмов

динамичному построению затребованного интерфейса пользователя в виде HTML-страницы. Можно добавить, что использование символических имен и дальнейшая их привязка к конкретным классам похожи на опосредованный вызов методов классов через методы интерфейсов (рис. 4.11).

#### **4.4. Взаимодействие элементов системы OFBIZ**

Три рассмотренные части системы OFBIZ естественным образом объединяются в единый каркас, позволяя создавать на своей основе цельные ERP-решения различной сложности.

Рассмотрим еще раз принципы работы ЦУС, ЦУЭД и контроллера, теперь в ходе их совместной работы. Уже упоминалось, что чаще всего решение на основе OFBIZ является распределенным Web-приложением (рис. 4.12)<sup>20</sup>.

В роли Front End (клиент) выступает обыкновенный Web-браузер, а Back End (сервер) состоит из какого-либо

<sup>20</sup> При этом архитектура системы OFBIZ позволяет просто создавать решения, в которых клиентом является специализированное приложение в виде Java-программы или Java-апплета. Эти приложения могут обращаться напрямую к ЦУС или ЦУЭД. Например, таким образом в системе OFBIZ реализована точка продажи (Point of Sale, POS).

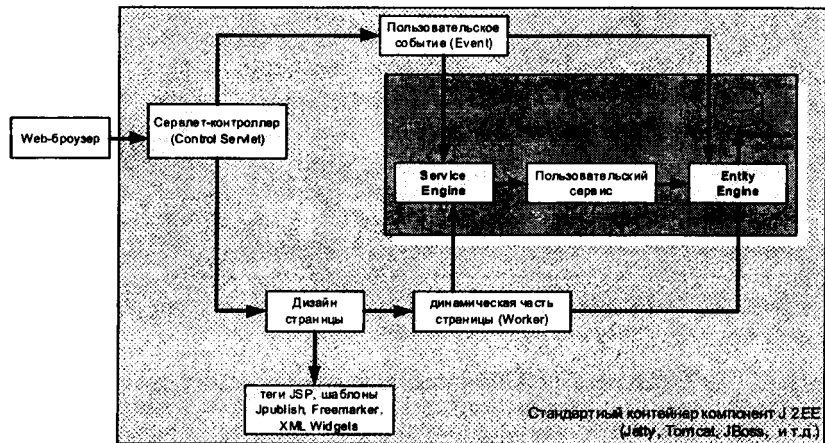


Рис. 4.12. Взаимодействие различных подсистем в процессе обработки пользовательских запросов

стандартного J2EE-контейнера с интегрированными в него компонентами OFBIZ.

Действия пользователя в виде запросов протокола HTTP по сети передаются на сервер, где за их обработку и подготовку ответа отвечает сервлет-контроллер.

В начале обработки, исходя из определения запроса в файле controller.xml, сервлет-контроллер может активизировать различные события (events) для взаимодействия с моделью. Обычно событием является вызов определенной службы OFBIZ. Через стандартный интерфейс сервлет-контроллер передает символическое имя службы и набор входных параметров в ЦУС, ожидая результата работы службы. ЦУС на основании анализа содержимого файла service.xml определяет конкретный способ реализации требуемой службы (Java-класс, скрипт на каком-либо языке интерпретируемого типа, вызов внешней системы и т.п.), проводит запуск нужных подсистем, инициализацию параметров и передачу результата работы в сервлет-контроллер.

При этом во время работы службы зачастую требуются различные элементы данных (например, имя и пароль для проверки на корректность введенной пользователем регист-



рациональной информации). Поиск и обеспечение доступа к нужным элементам данных — задача ЦУЭД. Через стандартный интерфейс из службы в ЦУЭД передаются все необходимые параметры (тип нужного элемента данных, условия поиска и т.п.) и по ним, а также по определению модели данных из конфигурационных xml-файлов (entitydef.xml), ЦУЭД автоматически генерирует запросы на языке SQL для установленной системы управления базами данных. Полученные в результате обработки SQL-запроса результаты преобразуются в экземпляры служебных Java-классов (обычно используется ассоциативный массив — Map) и возвращаются в службу<sup>21</sup>.

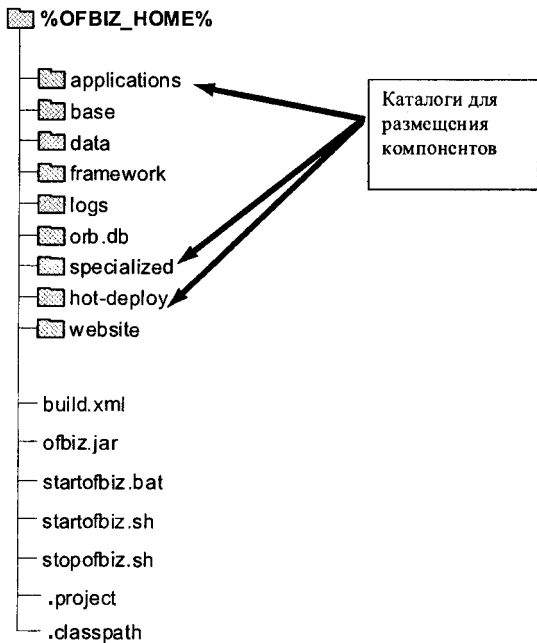
Кроме служб, во время активизации событий, сервлет-контроллер может непосредственно вызвать методы Java-классов, не являющихся службами. В этих классах разработчик также может свободно использовать возможности ЦУС и ЦУЭД для обращения к службам и работы с элементами данных.

По результатам активизации событий Сервлет-контроллер определяет необходимое представление и делегирует полномочия по генерации законченной HTML-страницы (или какого-либо иного объекта, например, отчета в виде динамического pdf-документа) ответственному обработчику. Во время генерации работает много алгоритмов преобразования и актуализации информации, в них также активно используются ЦУС и ЦУЭД для вызова служб и доступа к элементам данных.

Составная и динамичная внутренняя структура системы OFBIZ, сложные алгоритмы взаимодействия ее подсистем накладывают определенные требования на структуру

---

<sup>21</sup> Взаимодействие ЦУС и ЦУЭД может быть сложнее, если определены правила проверки непротиворечивости измененных элементов данных (так называемые ECAS — Event-Condition-Action Services). В этом случае изменению значений определенного элемента данных, вставке нового или удалению существующего экземпляра предшествует автоматический вызов службы из подходящего правила. Если служба «дает добро» на выполнение действия над элементом данных, то ЦУЭД доводит до конца работу с базой данных, выполняя требуемое действие. В противном случае, если служба обнаружила какие-то противоречия или внешние помехи для успешного завершения действий, ЦУЭД возвращает сообщение об ошибке, не изменяя содержимого базы данных.



**Рис. 4.13.** Структура каталогов верхнего уровня в системе OFBIZ

расширений, создаваемых независимыми разработчиками. Они должны быть хорошо знакомы со структурой и назначением стандартных каталогов системы OFBIZ, необходимым содержанием конфигурационных файлов и т.п.

Обычно в системе OFBIZ находится девять каталогов верхнего уровня и семь служебных файлов (рис. 4.13). Вся функциональность и информационная модель системы OFBIZ поделена между слабо связанными компонентами, каждый обычно содержит фрагмент информационной модели, набор служб, вспомогательных библиотек и определения нескольких пользовательских интерфейсов.

Стандартные компоненты, реализованные авторами системы OFBIZ, располагаются в каталоге

`%OFBIZ_HOME%/applications.`

Законченные решения или отдельные компоненты-расширения, создаваемые независимыми разработчиками, находятся в каталоге `%OFBIZ_HOME%/specialized`. Компоненты, которые достаточно изолированы, часто изменяются или находятся в процессе тестирования-отладки, могут размещаться в каталоге `%OFBIZ_HOME%/hot-deploy`.

Структура внутренних каталогов компонента системы OFBIZ в большой мере стандартизирована, поэтому знакомство только с одним из них даст полное представление о том, как устроены другие. На рис. 4.14 представлены наиболее важные каталоги и файлы, входящие в состав стандартного компонента `product`. Черный цвет используется для обозначения стандартных каталогов и стандартных файлов конфигурации, а серый цвет обозначает каталоги или файлы, специфичные для данного компонента.

Создаваемые независимыми разработчиками компоненты должны иметь похожую структуру, но при необходимости можно добавлять каталоги или файлы по своему усмотрению. Кроме того, OFBIZ разрешает ссылаться из конфигурационных файлов одного компонента к файлам другого компонента с помощью особого формата URI: `component:\\<относительный путь к файлу от начала файловой структуры компонента>`.

Нужно обратить внимание на то, что копирование файловой структуры нового компонента в

`%OFBIZ_HOME%/applications`

или

`%OFBIZ_HOME%/specialized`

не приводит к автоматическому подключению этого компонента. Для того чтобы определенные в компоненте элементы данных и службы стали доступны ЦУС и ЦУЭД, т.е. была проведена регистрация, необходимо добавить имя компонента в конфигурационный файл

`%OFBIZ_HOME%/base/conf/ofbiz-components.xml`

и перезагрузить систему.

Кратко выпишем назначение оставшихся каталогов верхнего уровня системы OFBIZ:

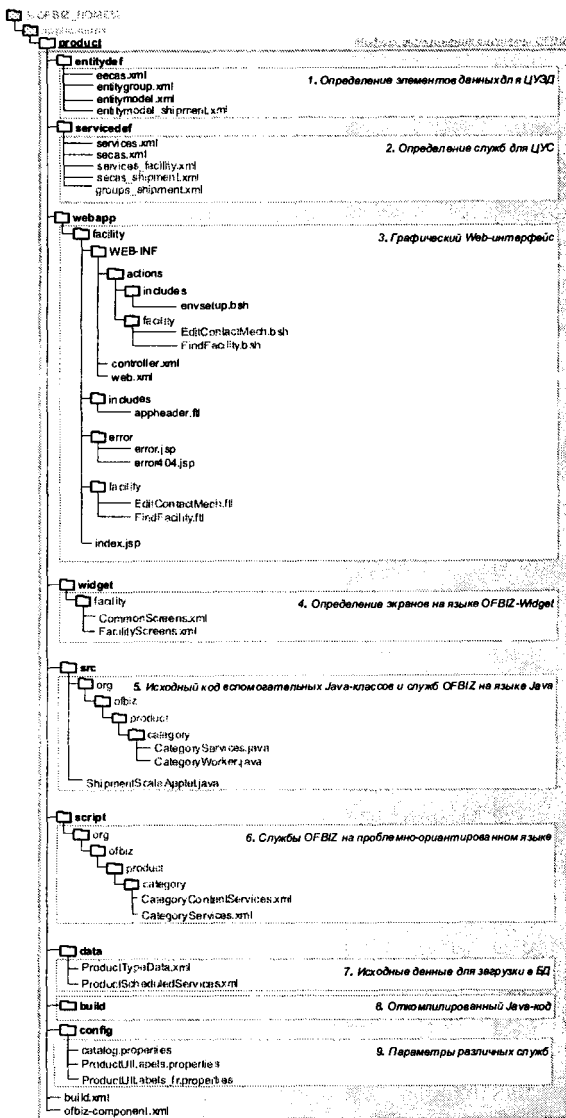


Рис. 4.14. Структура каталогов типичного компонента в системе OFBIZ

◆ **base** — основные файлы конфигурации, XML-схемы стандартных файлов конфигурации OFBIZ, библиотеки сторонних производителей;

◆ **framework** — исходный код каркаса системы OFBIZ, специфичные файлы конфигурации;

◆ **data** — хранилище данных и настройки для СУБД по умолчанию derby;

◆ **logs** — протоколы работы системы OFBIZ (логи);

◆ **orb.db** — протоколы доступа удаленных клиентов по стандартным протоколам;

◆ **website** — копия официального web-сайта OFBIZ.

После предварительного теоретического знакомства с возможностями и структурой системы OFBIZ можно приступить к практической работе.

## ***4.5. Инсталляция системы***

Для инсталляции системы OFBIZ предлагается использовать архив системы. Последнюю стабильную версию системы (на настоящий момент это `ofbiz-3.0.0.zip`) можно получить с Web-сайта

[http://sourceforge.net/project/show-files.php?group\\_id=27173](http://sourceforge.net/project/show-files.php?group_id=27173),

а также воспользоваться средствами сетевого доступа к рабочей версии исходного кода системы

<http://svn.ofbiz.org/>.

Разархивируйте полученный архив либо стандартную инсталляцию системы OFBIZ. Убедитесь, что верхний каталог системы называется `ofbiz`. Полный путь к этому каталогу от корня (включая и сам каталог `ofbiz`) мы будем по традиции называть `OFBIZ_HOME`. Для работы системы требуется JDK, начиная с версии 1.4, поэтому проверьте, что на вашей машине установлен полный JDK (JRE — недостаточен!).

Компиляцию исходного кода OFBIZ можно проводить либо с использованием инструментального средства ANT (аналога утилиты `make`), либо в среде разработки Eclipse. Здесь рассмотрим только последний вариант.

Для того чтобы компиляция OFBIZ в среде Eclipse была успешной, необходимо изменить использующуюся по

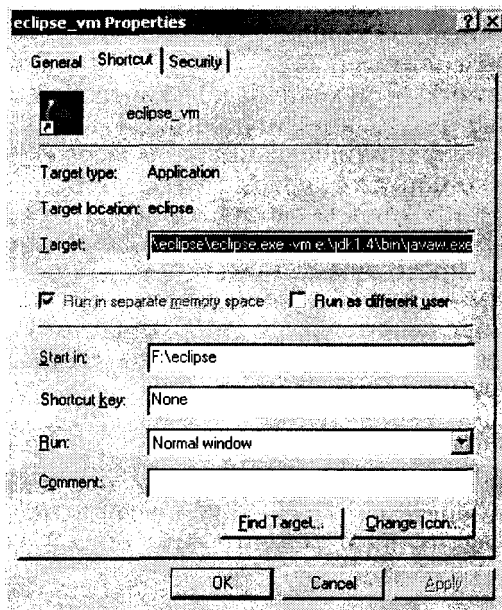


Рис. 4.15. В ОС Windows лучше всего создать особый ярлык для запуска Eclipse с ключом `-vm`.

умолчанию в Eclipse виртуальную java-машину на стандартную из поставки JDK 1.4.

Для этого нужно запускать Eclipse с ключом `-vm` <путь к стандартной программе `javaw` из JDK>. В операционной системе Windows лучше всего сразу сделать ярлык (Shortcut) с нужной строкой запуска (рис. 4.15), а для операционной системы Unix — создать отдельный скрипт для запуска Eclipse с ключом `-vm`.

Теперь запустите Eclipse с помощью созданного ярлыка или скрипта и закройте все проекты. Вам нужно будет создать новый проект с файлами OFBIZ. Это просто, потому что разработчики системы OFBIZ предоставляют вместе с исходным кодом и файл проекта `.project`, с помощью которого файловая структура OFBIZ автоматически импортируется в отдельный проект Eclipse.

В окне «Navigator» откройте контекстное меню (правая кнопка мыши) и выберите для выполнения команду «Import...». Появится диалоговое окно «Import,» в котором потребуется выбрать в качестве источника для импорта вариант «Existing Project into Workspace» и нажать на кнопку «Next>». На следующем шаге работы с диалогом нужно с помощью кнопки «Browse...» открыть стандартное окно выбора файлов и выделить файл «%OFBIZ\_НОЕ%\ . project». После этого закрыть окно и нажать в диалоге «Import» кнопку «Finish».

Результатом этих действий явится автоматическое создание нового проекта с именем «ofbiz» в окне «Navigator». Заметим, что в процессе импорта исходный код системы OFBIZ не будет физически переноситься в рабочее пространство Eclipse, а останется в каталоге «%OFBIZ\_HOME%\».

После импортирования проекта ofbiz можно провести компиляцию. Она по-прежнему будет выполняться с помощью инструментального средства ANT, но Eclipse предоставляет удобный интерфейс по управлению ходом его работы. Раскройте дерево файлов проекта ofbiz и выделите файл «build.xml». В нем содержатся инструкции для ANT, определяющие порядок построения готовой системы OFBIZ. Откройте контекстное меню на файле «build.xml» и выберите для выполнения команду «Run», а потом «1. Ant Build». Если все установки правильные, то в Eclipse активизируется окно «Console» и начнется процесс компиляции.

Для запуска системы OFBIZ также можно использовать файл «build.xml». Дважды щелкните по нему левой кнопкой мыши, и отобразится внутренняя структура с перечислением различных режимов построения готовой системы. Выберите из них директиву «run». Откройте контекстное меню на выделенной директиве и вызовите для выполнения команду «Run», а потом «1. Ant Build».

Откомпилированную систему OFBIZ можно запустить без Eclipse в операционной системе Windows или Unix. В %OFBIZ\_HOME% находится несколько скриптов для этого:

- ◆ «startofbiz.bat» — скрипт запуска для ОС Windows;
- ◆ «./startofbiz.sh» — скрипт запуска для ОС Linux/Unix;
- ◆ «./stopofbiz.sh» — скрипт остановки системы для ОС

Linux/Unix.

В ОС Windows остановка системы происходит после нажатия комбинации клавиш Ctrl-C в консольном окне с отладочной информацией OFBIZ (окно появляется автоматически после старта системы).

После запуска проверьте работоспособность системы, открыв в стандартном Web-браузере один из следующих адресов:

◆ <http://127.0.0.1:8080/ecommerce> — стартовая страница приложения ecommerce;

◆ <https://127.0.0.1:8443/webtools> — стартовая страница приложения WebTools;

◆ <https://127.0.0.1:8443/catalog> — стартовая страница приложения Catalog Manager.

Для работы в ряде приложений требуется предварительная регистрация. В системе OFBIZ существует предопределенный пользователь-администратор с такими характеристиками:

◆ username: «admin»;

◆ password: «ofbiz».

Если в ходе регистрации вам будет выдано предупреждение, что такого пользователя нет, то это означает, что внутренняя база данных системы еще не была проинициализирована. В таком случае нужно остановить систему и провести загрузку служебных данных. Как это сделать, сказано далее.

Чтобы полностью воспользоваться возможностями ЦУЭД, необходимо создать все таблицы в служебной базе данных и загрузить в них различные служебные данные (предопределенные параметры, типы данных, информацию о пользователях и т.п.).

Для выполнения этой операции нужно из командной строки, находясь в каталоге %OFBIZ\_HOME%, вызвать команду

```
java -jar ofbiz.jar -install 9900.
```

Параметр 9900 означает время тайм-аута в секундах при работе с базой данных. Его указывать не обязательно (тогда будет использоваться значение 7200 сек.).

Процессом инициализации и загрузки можно управлять, используя дополнительные ключи при запуске. Например:



- ◆ java -jar ofbiz.jar -install all 7200;
- ◆ java -jar ofbiz.jar -install seed,demo,ext 7200;
- ◆ java -jar ofbiz.jar -install seed.

Ключи определяют какой-либо фрагмент общей модели данных и имеют такое значение:

- ◆ seed — базовый набор данных и регистрационной информации о пользователях. После загрузки этого фрагмента можно регистрироваться в системе;
- ◆ demo — набор демонстрационных данных, которые нужны для показа функциональности приложений webstore, catalog, product, facility, content, accounting и manufacturing;
- ◆ ext — дополнительный набор данных, определяемый пользователем.

Для каждого компонента определено, откуда нужно брать данные для любого из возможных вариантов загрузки. Определения находятся в файле «ofbiz-component.xml». Например, для компонента «ecommerce» файл «components/ecommerce/ofbiz-component.xml» содержит такую информацию:

```

...
  <entity-resource type="data" reader-name="demo" loader="main"
location="data/DemoRentalProduct.xml"/>
  <entity-resource type="data" reader-name="ext" loader="main"
location="data/MyData.xml"/>
...

```

#### **4.6. Характеристика стандартных пользовательских приложений системы OFBIZ**

Все стандартные приложения системы OFBIZ используют единые правила по организации интерфейса в Web-браузере (рис. 4.16).

Графическое окно поделено по горизонтали на пять основных областей:

- ◆ Common Header — стандартная «шапка» для всех приложений, где обычно размещаются логотип системы, часы, выбор языка;

Стандартная панель инструментов. (Common AppBar), невидим, если пользователь не зарегистрирован

Панель сообщений (Message Field)

Основная секция интерфейса. (Section)

Рис. 4.16. Стандартные составляющие графического интерфейса приложения в системе OFBIZ

The screenshot shows a web browser window with the following components:

- Browser Title Bar:** OFBIZ: Party Manager: Find Party(s)
- Browser Menu Bar:** File Edit View Favorites Tools Help
- Browser Address Bar:** https://localhost:8443/partying/control/main?externalLoginKey=EL726236351546
- Common Header:** OPEN FOR BUSINESS OFBiz.org | Стандартная «шапка» приложения. (Common Header) | Welcome THE ADMINISTRATOR! 2005-02-01 15:55:01.757 | English (United States) | Set
- Common AppBar:** Accounting | Catalog | Content | Example | Facility | Manufacturing | Marketing | Стандартная панель инструментов. (Common AppBar)
- Party Manager Application:** Party Manager Application | Панель инструментов приложения (AppBar) | Main | Find | Create | Common | Views | Security | Logout
- Main Content Area:** Find Party | Contact Info:  None  Postal  Telecom  Other | Party ID:  | User Login:  | Last Name:  | First Name:  | Party Group Name:  | Role Type: Any Role Type | Show All Records | LookUp Party(s) | **Секция интерфейса приложения (Content Area)**
- Footer:** (c) 2004 The Open For Business Project - www.ofbiz.org | Powered By OFBiz | Стандартный шаблон приложения (Common Footer)

Рис. 4.17. Пример реализации различных составляющих интерфейса в приложении «Party» (вариант 1)

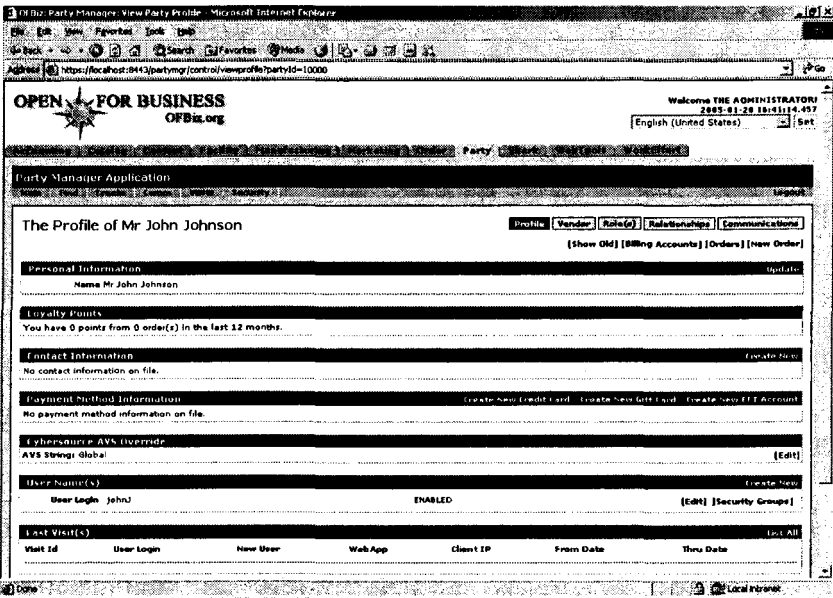


Рис. 4.18. Пример интерфейса в приложении «Party» (вариант 2)

- ◆ Common AppVar — стандартная панель инструментов для всех приложений, содержит закладки для доступа к стандартным приложениям системы OFBIZ. Стандартную панель можно запретить в конфигурации приложения;
- ◆ Application AppVar — панель инструментов, специфичная для каждого приложения, содержит команды, которые имеют смысл только в контексте данного приложения;
- ◆ Application Content Area — основная секция приложения, где реализуется весь специфичный интерфейс;
- ◆ Common Footer — стандартный «подвал» для всех приложений, содержащий ссылку на сайт разработчиков, соглашения об авторских правах и другую статичную информацию.

В рабочей области приложения находится специфичный пользовательский интерфейс (рис. 4.17, 4.18).

## 4.7. Контрольные задания к главе 4

1. *Индивидуальная работа.* Самостоятельно изучите основы работы в приложении «party» и создайте в нем 4—5 различных пользователей и групп с различными ролями.

2. *Индивидуальная работа.* По результатам перевода английского описания процесса производства (manufacturing, информацию возьмите с Web-страницы

<http://ofbizwiki.go-integral.com/Wiki.jsp?page=Manufacturing>

и самостоятельного изучения основы работы в приложении «manufacturing» создайте краткое руководство пользователя этого приложения.

3. *Индивидуальная работа.* Самостоятельно изучите основы работы в приложении «workeffort» и разработайте сценарий для наглядной демонстрации его основных возможностей. Продемонстрируйте результат преподавателю.

4. *Индивидуальная работа.* Продемонстрируйте возможность модификации данных системы из приложения «WebTools». Для этого создайте в приложении «facility» какой-нибудь элемент данных. Затем из приложения «WebTools» (раздел «Edit data Maintenance») выполните удаление этого элемента данных.

## Глава 5 \_\_\_\_\_

### Методы создания Web-интерфейсов в системе OFBIZ

Продолжим изучение механизмов реализации пользовательского интерфейса системы OFBIZ с момента, когда контроллер на основании содержимого элемента «request-mar» в файле controller.xml определил, какое представление требуется пользователю в ответ на запрос.

#### *5.1. Типы обработчиков*

В предыдущей главе было сказано, что в описании каждого представления, известного системе (тег <viewmar> в файле controller.xml), указывается кроме имени представления еще и его тип. Каждый тип представления связан с определенным обработчиком, которому делегируются полномочия по созданию представления.

Обычно обработчик — это Java-класс, в котором реализованы алгоритмы интерпретации какого-то языка описания интерфейса и преобразования по этому описанию исходных данных в определенный формат (HTML, PDF, XML и т.п.). В системе OFBIZ существует несколько заранее созданных обработчиков, связанных с предопределенными типами представлений (табл. 5.1).

Обычно связи для используемых обработчиков указываются в начале файла controller.xml.

#### *5.2. OFBIZ-Widgets*

Из всех перечисленных обработчиков мы подробно изучим только один — ответственный за подготовку HTML-

## Классы обработчиков OFBIZ

Тип представления (view type)	Класс обработчика, созданный в системе OFBIZ	Назначение
jsp	org.ofbiz.webapp.view.JspViewHandler	Динамичная подготовка HTML-страницы из JSP-страницы
ftl	org.ofbiz.webapp.ftl.FreeMarkerViewHandler	Динамичная подготовка HTML-страницы из описания на языке системы FreeMarker
http	org.ofbiz.webapp.view.HttpViewHandler	Передача пользователю содержимого HTML-страницы с удаленного сайта (т.е. реализация режима «прокси»)
screen	org.ofbiz.webapp.view.ScreenWidgetViewHandler	Динамичная подготовка HTML-страницы из описания на языке OFBIZ-Widgets
jpublish	org.ofbiz.webapp.view.JPublishViewHandler	Динамичная подготовка HTML-страницы из описания на языке системы JPublish
fop	org.ofbiz.webapp.view.FopPdfViewHandler	Динамичная подготовка PDF-документа в результате преобразования XML с помощью языка XSL-FO
screenfop	org.ofbiz.widget.screen.view.ScreenFopPdfViewHandler	Динамичная подготовка PDF-документа в результате преобразования XML с помощью языка XSL-FO
datavision	org.ofbiz.webapp.view.DataVisionViewHandler	Динамичная подготовка отчета в форме HTML-страницы из описания на языке системы Datavision
jasperreport spdf	org.ofbiz.webapp.view.JasperReportsPdfViewHandler	Динамичная подготовка отчета в форме PDF-документа из описания на языке системы JasperReport
jasperreport sxml	org.ofbiz.webapp.view.JasperReportsXmlViewHandler	Динамичная подготовка отчета в форме XML-документа из описания на языке системы JasperReport

страницы из описания на языке OFBIZ-Widgets (тип представления «screen»). Разработчики рассматривают этот язык наиболее перспективным и надеются на то, что по мере его развития можно будет одно и то же определение использовать для динамичной генерации различных фактических реализаций интерфейса: для Web-браузера (HTML), самостоятельного специализированного Java-приложения на настольном компьютере (Java Swing), мобильного устройства (WAP) и т.п.

В основе языка OFBIZ-Widgets лежит XML, поэтому описания представлений выражаются на нем в понятном текстовом, структурированном виде. При этом в OFBIZ-Widgets используются возможности языка XML по адресации произвольных структурных фрагментов, находящихся в других XML-файлах. Таким образом, можно разнести описание одного представления по различным файлам и использовать совместно общие фрагменты интерфейса.

### *5.2.1. Базовые принципы. Структура файла*

Законченным элементом описания одного представления в языке OFBIZ-Widgets является экран (screen). В том случае, когда представление преобразуется в формат HTML, то один экран соответствует одной HTML-странице. Каждый экран имеет уникальное имя, которое используется в атрибуте «name» элемента «view-map» в файле controller.xml. Кроме имени экрана в этом атрибуте указывается и путь к XML-файлу, в котором содержится описание экрана. В одном XML-файле можно определить произвольное количество различных экранов. Такой файл имеет структуру, указанную на рис. 5.1.

В файле описаний обязательно должен быть один элемент «screens», который может содержать внутри себя произвольное количество элементов «screen», каждый такой элемент соответствует определению одного экрана, которое фактически располагается внутри корневой секции («section») и поделено на последовательно идущие элементы: «condition», «actions», «widgets» и «fail-widgets». В элементе «condition» определяются логические условия, которые должны быть истинными, чтобы для подготовки экрана

использовались определения из элемента «widgets»<sup>22</sup>. Если логические условия являются ложными, то используются определения из «fail-widgets». В любом случае перед использованием выбранных определений выполняются действия, указанные в элементе «actions».

## screens

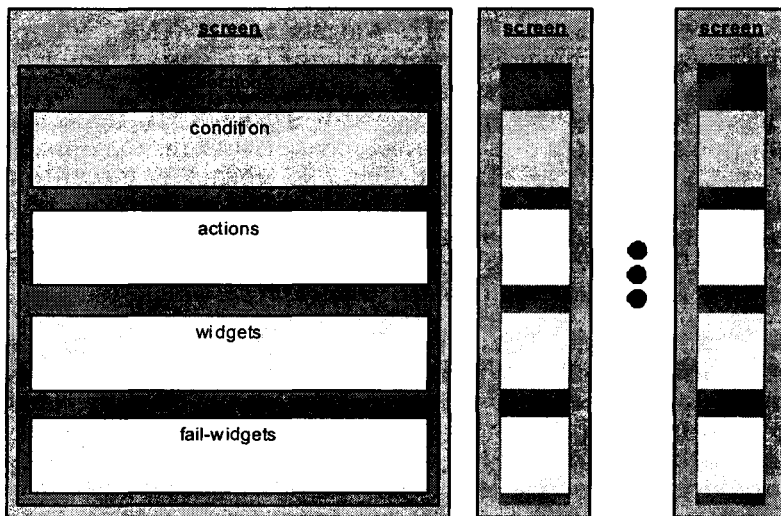


Рис. 5.1. Структура XML-файла с описанием экранов на языке OFBIZ-Widgets

Каждому упомянутому элементу описания соответствует одноименный XML-тег, а все правила по группировке этих тегов собраны в единую XML-схему

`%OFBIZ_HOME%\framework\widget\dtd\widget-screen.xsd`

Таким образом, XML-файл с описаниями экранов выглядит следующим образом:

<sup>22</sup> В свою очередь внутри элемента «widgets» или «fail-widgets» может находиться элемент «section».



```

<?xml version="1.0" encoding="UTF-8"?>
<screens>
  <screen name="left">
    <section>
      <condition> ... </condition>
      <actions> ... </ actions>
    <widgets>
      <section> ... </section>
    </widgets>
    <fail-widget> ... </fail-widget>
  </section>
</screen>

  <screen name="right">
    <section>
      <condition> ... </condition>
      <actions> ... </ actions>
    <widgets>
      <section> ... </section>
    </widgets>
    <fail-widget> ... </fail-widget>
  </section>
</screen>
  ...
</screens>

```

Изучение значения отдельных элементов описания экранов можно проводить на основе XML-схемы. Мы, однако, поступим иначе — за основу для детального описания основных элементов языка OFBIZ-Widgets будут взяты диаграммы классов языка UML, а фрагменты на XML будут использоваться в конкретных примерах. При построении UML-диаграмм приняты следующие особые правила:

- ◆ название класса на UML-диаграмме соответствует определенному XML-тегу;

- ◆ все поля класса являются публичными и соответствуют одноименному атрибуту XML-тега;

- ◆ тип полей задается через точку (большинство полей имеют тип String);

- ◆ допускается использование перечислимого типа поля, при этом в фигурных скобках задается набор разрешенных значений (например {«true», «false»}), а также после символа «=» указывается значение по умолчанию;

- ◆ идущий после определения типа поля символ «!» обозначает, что это поле соответствует обязательному атри-

буту XML-тега (необязательные атрибуты можно в XML-файле не использовать);

◆ XML-теги могут быть вложены друг в друга. На UML-диаграмме вложенность тега обозначается связью «композиция» (линия с ромбиком, внешнему тегу соответствует класс со стороны ромбика) и указанием множественности связи. Одинарная связь «композиция» с пунктирной линией означает, что вложенный тег может отсутствовать. Если несколько связей «композиция» выходят из одного ромбика, то это означает взаимоисключающие случаи вложения (только один тип тега может использоваться в каждом конкретном случае).

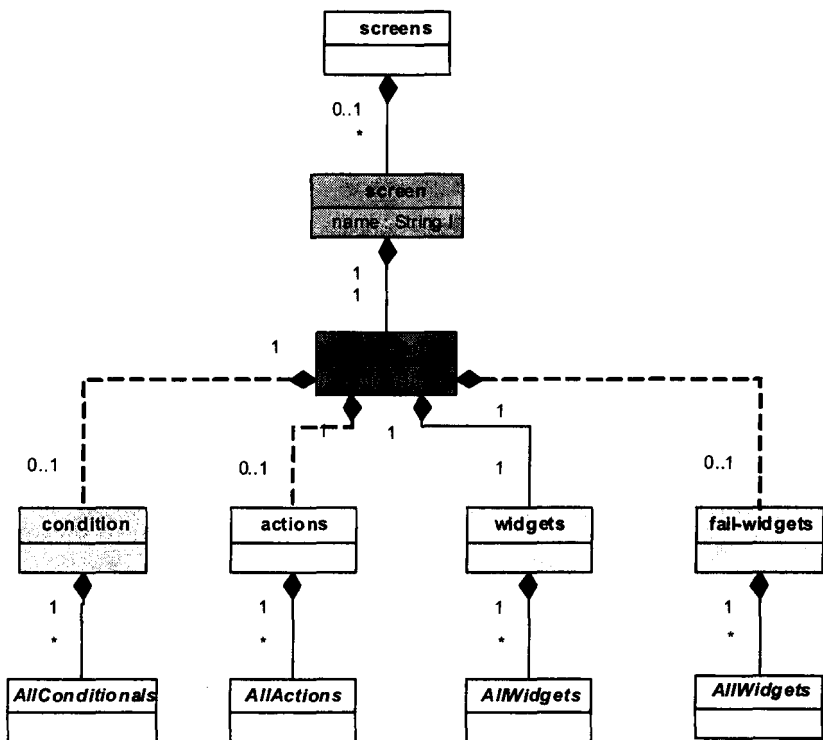


Рис. 5.2. Диаграмма классов верхнего уровня для описания экрана

Такой подход удобен, потому что диаграмма классов верхнего уровня практически полностью повторяет структуру XML-файла (рис. 5.2).

Заметьте, что особенностью структуры классов для описания экрана является активное использование абстрактных классов (их имена выделены наклонным шрифтом). От них происходит наследование различных элементов, реально используемых при определении содержания условий, действий и визуально видимых элементов. Это позволяет сделать язык OFBIZ-Widgets открытым для изменений и добавления новых выразительных возможностей.

Однако перед тем, как обратиться к рассмотрению наследников абстрактных классов AllConditionals, AllActions и AllWidgets, нужно сначала узнать о модели данных.

### 5.2.2. Модель данных

Важной составляющей языка OFBIZ-Widgets является модель данных. Она предоставляет во время подготовки интерфейса пользователя удобный доступ к содержимому и методам различных внутренних Java-объектов системы OFBIZ. Особенностью реализации модели данных в языке OFBIZ-Widgets является то, что вместо привычного доступа к экземплярам различных Java-объектов используется более абстрактный механизм работы с совокупностью взаимосвязанных полей-переменных, позволяющий единообразно обращаться к значениям различных видов объектов без детального знания особенностей их реализации.

С точки зрения разработчика описания интерфейса на языке OFBIZ-Widgets модель данных подобна дереву (см. рис. 5.3).

Поля-переменные (fields), которые являются листьями дерева и хранят единственное значение, называются скалярами. Значения скаляров могут иметь один из следующих типов: String; Number; Boolean; DateTime и др.

Поля-переменные (fields), которые соответствуют внутренним вершинам дерева, являются контейнерами и содержат в себе подчиненные поля-переменные (например, поле с именем «latestProduct» является контейнером). Поля-контейнеры обычно представляют из себя хеш или ассоциа-

```

(root)
|
+- user = "Big Joe"
|
+- latestProduct
|   |
|   +- url = "products/greenmouse.html"
|   |
|   +- name = "green mouse"
|
+- weeks
|   |
|   +- [0]
|       |
|       +- price = 100
|   |
|   +- [1]
|       |
|       +- price = 200

```

Рис. 5.3. Пример модели данных в языке OFBIZ-Widgets

тивный массив. В нем все данные хранятся в виде пар «имя поля»—«значение поля», а программисту предоставлены методы для добавления новых пар (т.е. создания нового подчиненного поля), получения значения поля по его имени, удалению пар. Если из ассоциативного массива запрашивается значение неизвестного поля, то возвращается пустой результат. Для программной реализации ассоциативного массива в системе OFBIZ используется стандартный Java-класс с интерфейсом Map. Еще один тип полей-переменных, являющихся контейнерами, — это последовательность (sequence). Переменная этого типа похожа на хеш и содержит несколько упорядоченных подчиненных полей. Однако их символические имена в последовательности не сохраняются, поэтому доступ к подчиненным полям осуществляется с помощью порядкового числового индекса, как в массиве.

В модели данных OFBIZ для доступа к произвольному полю-переменной, являющемуся либо хешем, либо скаляром, необходимо указать полный путь от корня (не включая саму строку «root»), последовательно перечисляя имена

родительских полей-переменных и отделяя каждое такое имя символом точки. Например, для того чтобы обратиться к полю «name» из рис. 5.3, нужно составить такой путь: latestProduct.name .

Для доступа к подчиненному полю последовательности нужно указать соответствующий числовой индекс в квадратных скобках. При этом нужно запомнить, что значение индексов начинается с нуля. Таким образом, для того чтобы получить доступ к полю-переменной «price» из рис. 5.3, находящемуся в последовательности «weeks» на второй позиции, нужно использовать такое выражение:

weeks[1].price

Последовательности являются динамическими структурами — можно добавлять новые подчиненные поля в конец или между существующими. Для этого используется синтаксис [+ind], где ind — числовой индекс подчиненного поля, после которого нужно добавить новое поле.

Благодаря всем этим особенностям есть возможность просто и безопасно динамично вводить и использовать необходимые пользовательские поля-переменные в описании экранов. Кроме того, в форме полей с особыми именами разработчику описания экрана доступна разнообразная служебная информация о конфигурации системы, переданных из Web-браузера параметрах и т.п. Наиболее важные предопределенные служебные поля приведены в табл. 5.2.

Поля-переменные, входящие в состав модели данных, в языке OFBIZ-Widgets можно использовать при описании экранов двумя способами.

Подстановка значения поля в содержимое XML-файла. Для подстановки используется формат

\${<путь\_к\_переменной>}

Например:

```
<widgets>
<include-screen name="${detailScreen}"/>
  <label style="head1">${uiLabelMap.AccountingFixedAssets}</label>
</widgets>
```

1. Специальный XML-тег, определяющий некоторую операцию с полем (инициализация, проверка и т.п.). Например:

## Служебные поля

Имя поля модели данных	Тип	Назначение
1	2	3
globalContext	Hash	содержит поля модели данных с глобальной областью видимости, в том числе и все переменные, определенные ниже
context	Hash	содержит поля модели данных с локальной областью видимости, в том числе и все переменные, определенные ниже
screens	javaClass	предоставляет доступ к методам интерфейса <code>Renderer</code>
parameters	Hash	содержит параметры исходного HTTP-запроса и дополнительную информацию
Delegator	javaClass	предоставляет доступ к методам интерфейса <code>Delegator</code>
dispatcher	javaClass	предоставляет доступ к методам интерфейса <code>Dispatcher</code>
security	javaClass	предоставляет доступ к методам интерфейса <code>Security</code>
nullField	String	наименование пустого слова
userLogin	String	текущий код идентификации зарегистрированного пользователя
autoUserLogin	Boolean	автоматически перенаправлять пользователя на страницу регистрации ?
person	String	имя зарегистрированного пользователя
partyGroup	String	текущая группа, к которой принадлежит зарегистрированный пользователь
request	javaClass	предоставляет доступ к методам интерфейса <code>Request</code>
response	javaClass	предоставляет доступ к методам интерфейса <code>Response</code>
session	javaClass	предоставляет доступ к методам интерфейса <code>Session</code>
application --- ServletContext	Hash	набор параметров, определяющий контекст обработки в сервлете

*Продолжение таблицы 5.2*

1	2	3
rootDir	String	локальный файловый путь к каталогу, где размещаются файлы web-приложения текущего компонента системы OFBIZ
webSiteId	String	текущий идентификатор логического Web-сайта в системе OFBIZ
https	Boolean	проходит ли работа в защищенном режиме?
locale	String	текущая локализация
available Locales	Sequence	последовательность строк, обозначающих различные доступные локализации
controlPath	String	частичный путь (URL без сервера) к сервлету-контроллеру текущего компонента
contextRoot	String	локальный файловый путь к каталогу текущего компонента системы OFBIZ
serverRoot	String	адрес сервера системы OFBIZ (URL)
checkLoginUrl	String	частичный путь (URL без сервера) к представлению для проверки пароля
isError	Boolean	была ли страница передана для сообщения об ошибке
nowTimestamp	String	текущее системное время и дата
request Attributes	Hash	атрибуты формы, полученные из HTTP-запроса

```
<set field="viewIndex" from-field="requestParameters.VIEW_INDEX"
type="Integer"/>
```

Первый способ может использоваться в любом месте XML-файла, а второй — только в разделах условий или действий.

Нужно отметить, что поля модели данных в системе OFBIZ обладают глобальной областью видимости. Это означает, что к одному и тому же полю можно обращаться по одинаковому имени из различных разделов описания экрана («condition», «actions», «widgets», «fail-widgets»).

### 5.2.3. Содержимое разделов описания

Теперь можно вернуться к рассмотрению элементов описания экранов — наследников виртуальных классов из диаграммы на рис. 5.2. На рис. 5.4 представлена диаграмма, в которой собраны элементы для определения условий — наследники класса AllConditionals.

Значение основных элементов (XML-тегов) дано в табл. 5.3.

Например, на рис. 5.5 показан фрагмент XML-файла, определяющий такое условие: экран может быть показан пользователю лишь в том случае, когда он имеет разрешение на просмотр экранов («\_VIEW»), относящихся к приложению «ACCOUNTING», или имеет разрешение на просмотр экранов, относящихся к приложению «PARTY».

Таблица 5.3

**Назначение основных элементов**

Название	Назначение
1	2
if-compare	Производится сравнение значения поля из модели данных с какой-то константой
if-compare-field	Производится сравнение значения одного поля из модели данных со значением какого-то другого поля из модели данных
if-empty	Определяется существование в модели данных поля с указанным именем
if-entity-permission	Во время подготовки представления проверяется, разрешено ли обращаться к определенному элементу данных из ЦУЭД
if-has-permission	Во время подготовки представления проверяется, есть ли у текущего пользователя права доступа к данному представлению для выполнения определенной операции
if-regex	Проверяется, содержит ли значение поля модели данных с указанным именем определенное значение, заданное регулярным выражением
if-validate-method	Для указанного поля модели данных вызывается с целью проверки определенный статический метод Java-класса
and	Двухместный логический оператор И



Продолжение таблицы 5.3

1	2
not	Одноместный логический оператор отрицания
or	Двухместный логический оператор ИЛИ
xor	Двухместный логический оператор Исключающее-ИЛИ
set	Присвоение определенного значения указанному полю из модели данных. Источником значения может быть константа или другое поле модели данных
property-map	Осуществляется чтение из указанного файла Java-свойств и преобразование его значений в модель данных. Напомним, что в текстовом файле Java-свойств находятся строки формата «имя_свойства=значение_свойства». Имена свойств обычно выглядят как слова, разделенные точками, поэтому весь набор свойств преобразуется в модель данных OFBIZ естественным образом. Отметим, что по правилам системы OFBIZ используемый файл свойств должен располагаться в каталоге config данного компонента
property-to-field	Присвоение определенного значения указанному полю из модели данных. Источником значения является одноименный ресурс из указанного файла java-свойств. По правилам системы OFBIZ используемый файл свойств должен располагаться в каталоге config данного компонента.
script	Вызывается внешний скрипт на определенном языке интерпретируемого типа
service	Вызывается определенная служба системы OFBIZ
entity-one	Из ЦУЭД выбирается ровно один экземпляр Элемента Данных определенного типа по критерию равенства ключевых атрибутов с определенными полями из модели данных. Выбранный экземпляр помещается в модель данных OFBIZ как ассоциативный массив и доступен в определении экранов и форм
entity-and	Из ЦУЭД выбираются все экземпляры Элементов Данных определенного типа по критерию равенства ключевых атрибутов с определенными полями из модели данных. Выбранные экземпляры помещаются в список модели данных OFBIZ и доступны в определении экранов и форм
entity-condition	Из ЦУЭД выбираются все экземпляры Элементов Данных с использованием сложного критерия поиска.

1	2
	<p>Выбранные экземпляры помещаются в список (list) модели данных OFBIZ и доступны в определении экранов и форм.</p> <p>Для определения сложного критерия поиска могут использоваться различные операции сравнения ключевых и неключевых атрибутов Элемента Данных с константами или полями модели данных (меньше, больше, равно) и логические связки И, ИЛИ. Кроме того, можно определять, какие атрибуты Элемента Данных будут использованы для формирования списка, порядок сортировки экземпляров и другие параметры выборки</p>
label	Строчка текста
container	Контейнер для других элементов интерфейса. Может использоваться для определения рамки вокруг какой-то части окна
image	Графическое изображение
link	Интерактивная ссылка на другой экран (например, на другую HTML-страницу внешнего сайта)
content	Позволяет включать в состав экрана какую-то информацию, находящуюся под управлением подсистемы манипулирования содержанием (OFBIZ Content Manager) <sup>23</sup>
sub-content	Позволяет включать в состав экрана какую-то информацию, находящуюся под управлением подсистемы манипулирования содержанием (OFBIZ content Manager)
include-screen	Включает в состав экрана другой экран
include-form	Включает в состав экрана форму
include-menu	Включает в состав экрана меню
include-tree	Включает в состав экрана дерево
decorator-screen	Определяет, что экран включен в объемлющий экран-декоратор, используется шаблон «декоратор» (см. далее)
decorator-section	Определяет, в какую секцию экрана-декоратора включается экран, используется шаблон «декоратор» (см. далее)
decorator-section-include	Определяет секцию экрана-декоратора, используется шаблон «декоратор» (см. далее)
iterate-section	Определяет секцию в виде таблицы

<sup>23</sup> Понятия Content, SubContent, Content Manager и общие принципы управления информацией будут разобраны в приложениях к этой главе.

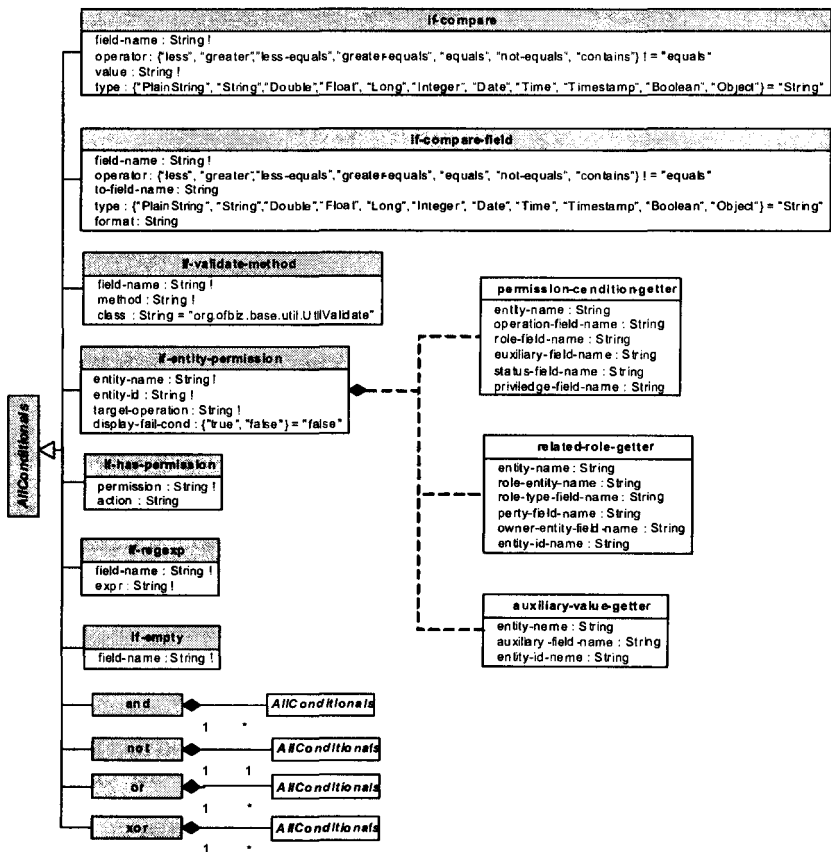


Рис. 5.4. Элементы для определения условий

```

<condition>
  <or>
    <if-has-permission permission="ACCOUNTING" action="_VIEW"/>
    <if-has-permission permission="PARTY" action="_VIEW"/>
  </or>
</condition>

```

Рис. 5.5. Использование логических операторов

```

<widgets>
  <section>
    <condition>
      <not><if-empty field-name="productCategory"/></not>
    </condition>
    <widgets>
      <include-screen name="{detailScreen}"/>
    </widgets>
    <fail-widgets>
      <label style="head2"> Product Category was not defined
      ${uiLabelMap.ProductCategoryNotFoundforCategoryId} ${productCategoryId}!
      </label>
    </fail-widgets>
  </section>
</widgets>

```

Рис. 5.6. Использование условий при подготовке экрана

Еще один пример. В нем содержимое экрана показывается пользователю в том случае, если определено поле-переменная с именем «productCategory». В противном случае отображается сообщение об ошибке из секции «fail-widgets» (рис. 5.6).

Возможные действия для манипуляции с моделью данных во время подготовки экрана определены на UML-диаграмме в рис. 5.7.

На рис. 5.8 тег <set> инициализирует поле-переменную определенным значением.

На рис. 5.9 приведен пример чтения названий (текстовых заголовков) элементов интерфейса из файла свойств. Сначала читаются названия, специфичные для приложения «Accounting» из файла свойств. Затем, наконец, можно проинициализировать поле «layoutSettings.companyName».

На рис. 5.10 приведен пример вызова службы.

Мы не будем подробно рассматривать те действия, которые связаны с взаимодействием с ЦУЭД (вернемся к ним позже). Поэтому без дополнительных комментариев на рис. 5.11 приведем примеры.

### *Элементы интерфейса*

Переходим к описанию элементов, определяющих визуально доступные части интерфейса. Соответствующая UML-диаграмма представлена на рис. 5.12.

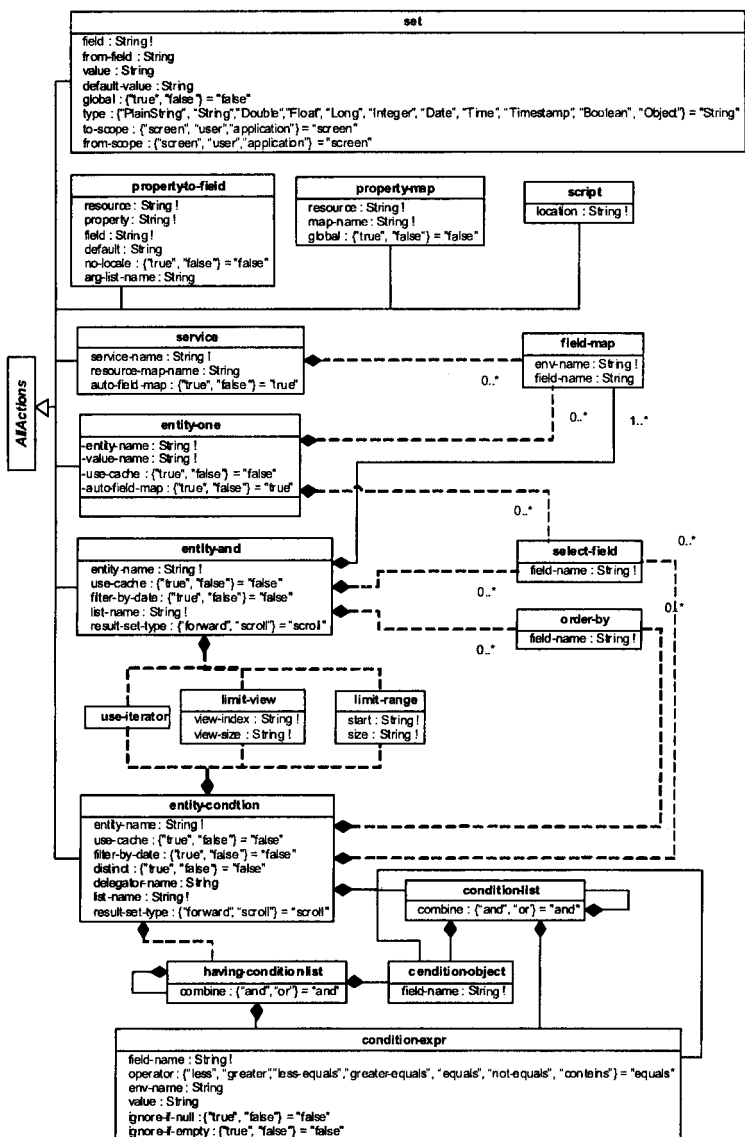


Рис. 5.7. Элементы для определения действий

```
<set field="viewIndex" from-field="requestParameters.VIEW_INDEX" type="Integer"/>
<set field="viewSize" from-field="requestParameters.VIEW_SIZE" type="Integer" default-
value="20"/>
```

**Рис. 5.8.** Использование тега `<set>`

```
<property-map resource="AccountingUiLabels" map-name="uiLabelMap" global="true"/>
<property-map resource="CommonUiLabels" map-name="uiLabelMap" global="true"/>
<set field="layoutSettings.companyName" from-
field="uiLabelMap.AccountingCompanyName" global="true"/>
```

**Рис. 5.9.** Использование тега `<property-map>`

```
<service service-name="urlEncodeArgs" result-map-name="result">
  <field-map env-name="requestParameters" field-name="mapIn"/>
</service>
```

**Рис. 5.10.** Использование тега `<service>`

```
<entity-one entity-name="GIAccount" value-name="glAccount"/>
<entity-condition entity-name="GIAccount" list-name="entityList" use-cache="true" >
  <condition-expr field-name="glAccountId" operator="greater" value="0" />
  <limit-range start="0" size="10" />
</entity-condition>
<entity-condition entity-name="FixedAssetProduct" list-name="fixedAssetProducts" >
  <condition-expr field-name="fixedAssetId" operator="equals" value="{fixedAssetId}"
/>
  <order-by field-name="productId"/>
  <order-by field-name="fixedAssetProductTypeId"/>
  <order-by field-name="fromDate"/>
</entity-condition>
```

**Рис. 5.11.** Примеры действий с ЦУЭД

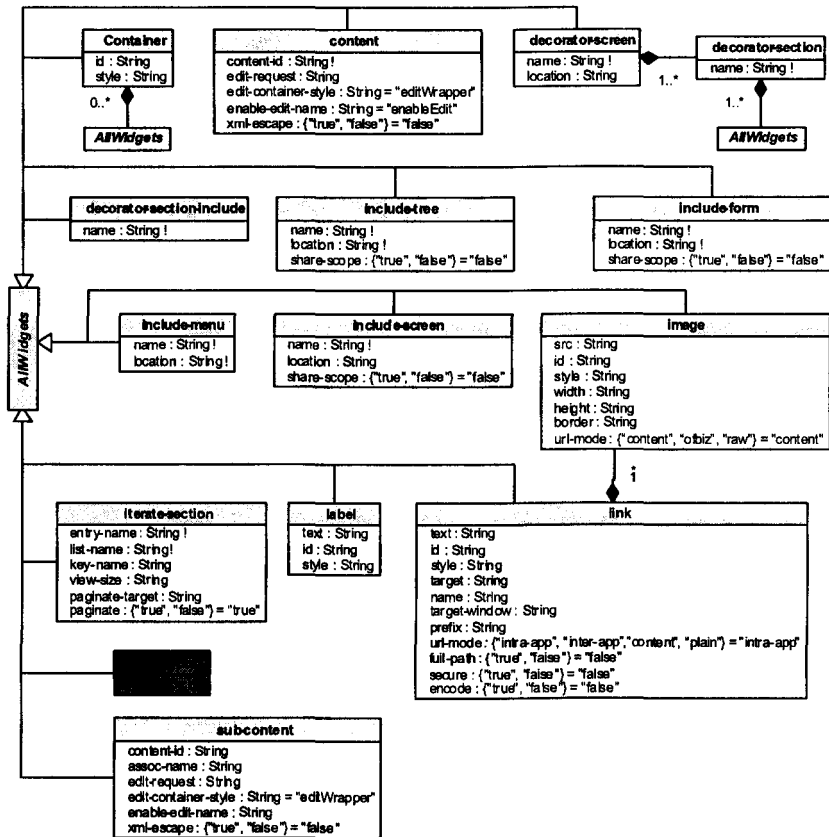


Рис. 5.12. Элементы для определения интерфейса

### 5.3. Стандартные составные элементы

В реализации интерфейса на любой платформе можно выделить несколько общеупотребительных составных элементов, например, формы, деревья и меню. В языке OFBiz-Widgets для определения таких составных элементов используются соответствующие составные XML-теги.

### 5.3.1. Формы

Форма предоставляет удобный интерфейс для просмотра или редактирования элементов данных, находящихся под управлением ЦУЭД, результатов работы служб, а также ввода различной информации от пользователя с последующей передачей ее на обработку и вызова различных команд. Визуально форма выглядит как набор именованных полей или таблица. Кроме того, на форме могут располагаться различные кнопки, списки выбора, переключатели и другие графические элементы интерфейса (рис. 5.13, 5.14).

Каждая отдельная форма включается в описание экрана с помощью тега `<include-form>`. В нем указывается символическое имя формы и ссылка на XML-файл, в котором хранится полное определение структуры формы.

**Find Party**
**Lookup Party(s)**

**Contact Info:**     None     Postal     Telecom     Other

**Party ID:**   

**User Login:**   

**Last Name:**   

**First Name:**   

**Party Group Name:**   

**Role Type:**   

---

Show All Records

Edit	Contact Type Id	Owner Contact Id	Data Resource Id	Template Data Resource Id	Data Source Id	Status Id	Privilege Enum Id	Service Name	Contact Name
TEMPLATE_MASTER	TREE_ROOT								
STDWRAP001	DOCUMENT		STDWRAP001						Standard SubContent Wrapper
TP_TECHNICAL	PLACEHOLDER								Technical
TP_LEGAL	PLACEHOLDER								Legal
TP_BUSINESS	PLACEHOLDER								Business
LEFTBAR	DOCUMENT		LEFTBAR						Left Bar

Рис. 5.13. Примеры форм в системе OFBIZ



Синтаксис определения формы задается особой XML-схемой, которая располагается в файле

`%OFBIZ_HOME%\framework\widget\dtd\widget-form.xsd`.

Как и в случае с определениями экранов, в одном XML-файле может располагаться несколько форм внутри главного XML-тега `<forms>`. Структура элементов описания формы представлена на рис. 5.15.

Приведем несколько советов и правил использования форм (детальные указания следует взять из документации по OFBIZ).

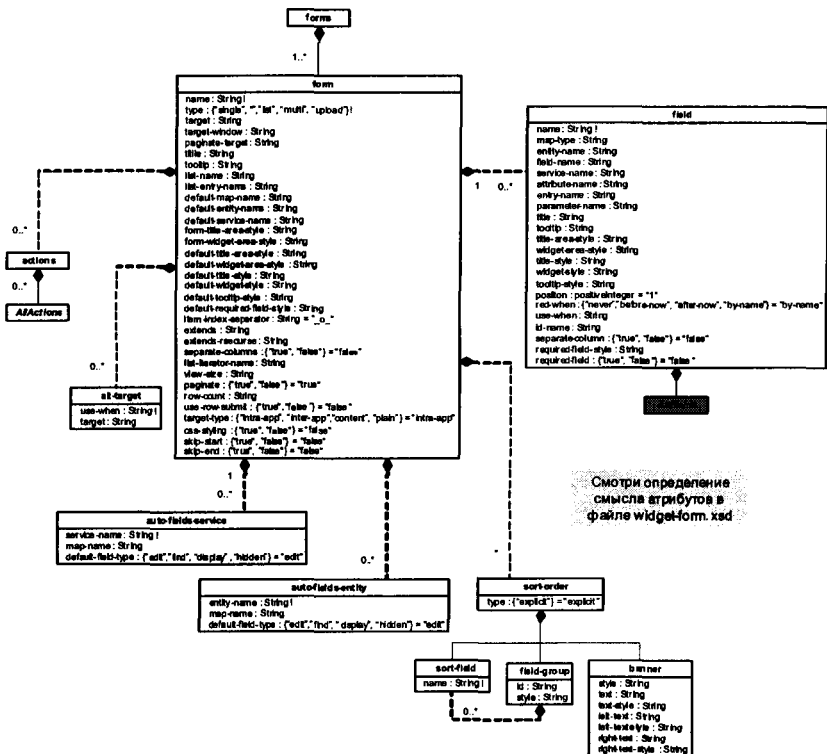


Рис. 5.14. Элементы верхнего уровня для описания форм в языке OFBIZ-Widgets

1. В системе OFBIZ формы бывают четырех типов: простые (single), в виде таблицы (list), мульти-формы (multi) и формы для загрузки на сервер файлов (upload). Формы типа «single» позволяют за раз просматривать/редактировать один элемент данных из ЦУЭД или один ассоциативный массив из модели данных. Формы типа «list» одновременно представляют в виде таблицы множество элементов данных из ЦУЭД (или последовательность одинаковых ассоциативных массивов из модели данных). В такой таблице за один раз можно редактировать и передавать на обработку данные, относящиеся только к одному элементу данных (или одному ассоциативному массиву). Формы типа «multi» также представляют значения нескольких элементов данных (или массивов) в виде таблицы, но вдобавок обеспечивают уникальную идентификацию каждой строчки таблицы. Это позволяет передавать на обработку несколько измененных строк таблицы одновременно.

2. В системе OFBIZ разработчик может использовать разнообразные источники данных для заполнения значениями полей для форм любых типов; при подготовке экрана данные могут автоматически помещаться в поля формы из:

- ♦ определенного ассоциативного массива из модели данных;

- ♦ одного (для формы типа single) или нескольких экземпляров элементов данных, полученных из ЦУЭД в результате поиска по определенным критериям;

- ♦ выходных аргументов службы.

3. Особенностью реализации форм в системе OFBIZ является возможность автоматической синхронизации перечня полей в форме со структурой определенного ассоциативного массива из модели данных, с набором полей в каком-либо элементе данных, или со списком входных/выходных параметров какой-либо службы. Таким образом, при изменении в структуре ассоциативного массива, в элементе данных или в перечне (типах) аргументов службы никаких переделок в связанных с ними формах делать не приходится.

4. Если для заполнения полей формы типа «single» используется ассоциативный массив модели данных, то его имя указывается в параметре default-map-name тега <form>. Перечень полей в форме будет синхронизирован с полями

5. Если для заполнения полей формы типа «list» используется последовательность ассоциативных массивов модели данных, то имя последовательности указывается в параметре list-name тега <form>. Перечень полей в форме будет синхронизирован с полями ассоциативного массива из указанной последовательности.

6. Если для заполнения полей формы используется значения элемента данных ЦУЭД, имя этого элемента данных указывается в параметре entity-name вложенного

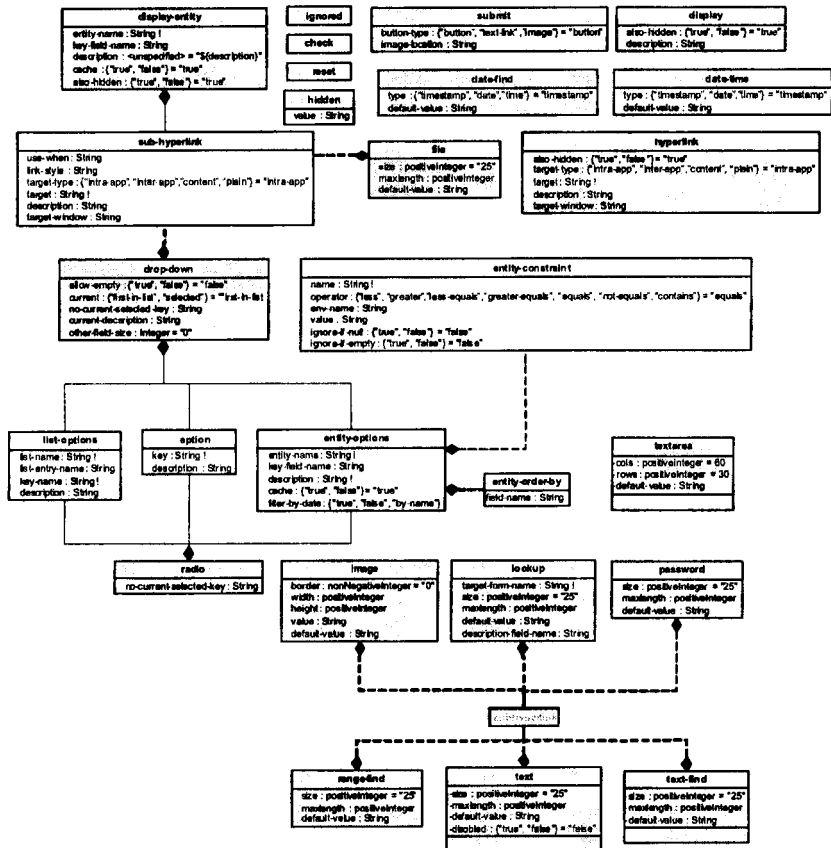


Рис. 5.15. Элементы для определения содержимого формы

тега <auto-fields-entony>. Способ визуального представления данных определяется значением атрибута default-field-type.

7. Если перечень полей формы соответствует списку параметров определенной службы, то имя этой службы указывается в параметре service-name вложенного тега <auto-fields-service>. Способ визуального представления данных определяется значением атрибута default-field-type.

8. Данные из формы могут пересылаться для обработки различным обработчикам в зависимости от содержимого модели данных. Для этого в составе формы может использоваться вложенный тег <alt-target>, который определяет альтернативный обработчик (POST target). Он будет использоваться в том случае, если условие, определенное в атрибуте use-when, истинно на момент подготовки HTML-страницы из описания на языке OFBIZ-widget.

9. Для любого поля можно указать условия включения в состав формы. Условия определяются значением атрибута use-when вложенного тега field. Синтаксис условного выражения в атрибуте use-when должен соответствовать требованиям языка Java и XML. Например, допустим такой вариант:

```
use-when="associationToEdit.get("productCategoryIdTo")!=null".
```

10. Для принудительного представления поля формы в виде отдельной колонки таблицы можно использовать атрибут separate-field тега <field>.

11. Язык OFBIZ-Widgets содержит несколько вложенных тегов для определения внешнего вида полей формы: тег <display> определяет представление значения в виде текстовой строки; тег <date-time> определяет представление значения в виде строки в формате дата/время; тег <edit> определяет представление значения в виде многострочной текстовой области; тег <submit> служит для размещения на форме особой кнопки «Submit»; тег <hyper-link> позволяет в составе формы создать гипер-ссылку на другую страницу или иной ресурс; тег <drop-down> определяет список из группы элементов; тег <hidden> определяет невидимое поле.

12. Чтобы указать предопределенные значения в списке выбора, заданном тегом <drop-down>, необходимо использовать вложенный тег <option>.

13. Для того чтобы в качестве значений списка использовать экземпляры элементов данных ЦУЭД, необходимо

воспользоваться вложенным тегом `<entity-options>`. Он позволяет указать тип элемента данных и его ключевое поле. В таком случае список выбора на HTML-странице будет содержать значения ключевых полей всех существующих экземпляров (значения ключевых полей будут помещены в атрибут `value` HTML-тега `<option>`). Дополнительно можно указать описание (`description`), которое будет отображаться для пользователя. В значение атрибута `description` можно подставлять элементы модели данных OFBIZ: `#{description}`.

14. Тег `<sort-order>` определяет только порядок колонок в таблице, но не порядок строк в таблице.

15. Использование тега `<display-entity>` вместо тега `<display>` позволяет по заданному имени ключевого поля отобразить на форме значения полей связанного элемента данных. Например, выражение:

```
<field name="productPriceTypeId">  
<display-entity entity-name="ProductPrice/">  
</field>
```

позволяет включить в текущую форму значения всех полей связанного экземпляра элемента данных с типом `ProductPrice`. Критерием поиска нужного экземпляра является значение поля `productPriceTypeId` в текущем экземпляре элемента данных.

16. Для определения содержимого формы типа «list» также может использоваться поле модели данных OFBIZ особым типом `EntityListIterator`. По сути это поле является экземпляром `java`-класса `EntityListIterator` и возвращает последовательно экземпляры элементов данных ЦУЭД по определенному логическому условию (так называемый итератор). Имя используемого поля-итератора задается в атрибуте `list-iterator-name` тега `<form>`. Создание экземпляра итератора и включение его в состав модели данных OFBIZ происходит в результате вызова какой-нибудь службы. Следующий пример показывает, как можно получить данные в форме типа «list», используя лишь средства языка OFBIZ-Widget:

```
<form name="listCampaigns" type="list" target="" default-entity-  
name="MarketingCampaign"  
list-iterator-name="listIt"
```

```

default-title-style="tableheadtext" default-widget-style="inputBox"
default-tooltip-style="tabletext" paginate-target="" view-size="20">
<actions>
  <service service-name="performFind" list-iterator-name="listIt">
    <field-map field-name="inputFields" env-name="requestParameters"/>
    <field-map field-name="entityName" env-name="entityName"/>
  </service>
</actions>
<field ...
...
</form>

```

При этом значение атрибута `list-iterator-name` в теге `<form>` является именем соответствующего поля в модели данных OFBIZ (`list-iterator-name = «listIt?»`). Это поле содержит экземпляр итератора с типом `EntityListIterator`. Он был создан в результате работы службы `performFind`

(`org.ofbiz.com-mon.FindServices`)

и передан этой службой в выходном аргументе с именем `«listIt»`. Служба `performFind` в качестве входных аргументов принимает на вход критерии поиска (ассоциативный массив `inputFields`) и тип элемента данных ЦУЭД (скаляр `entityName`). Ассоциативный массив `«requestParameters»`, содержащий критерии поиска для работы службы, был заранее инициализирован в какой-то другой форме (например, `«findXxxx»`). Элемент модели данных с именем `«entityName»` был заранее инициализирован в блоке `<actions>` описания экрана.

Конечно, существуют и ограничения при работе с формами на языке OFBIZ-Widget, в частности:

1) нет возможности провести сортировку строк таблицы списка в форме типа `«list»` на этапе отображения данных. Сортировку можно выполнить лишь на этапе подготовки данных в службе или на этапе выборки определенных экземпляров элементов данных ЦУЭД в соответствии с параметрами тегов `<entity-and>` или `<entity-condition>`;

2) нельзя контролировать ширину полей типа `<display>`. Для определения значения поля формы можно использовать два взаимоисключающих атрибута: `env-name` и `value`. Атрибут `env-name` содержит имя поля модели данных, а атрибут `value` содержит текстовую константу. Однако при

использовании атрибута `value` разрешается подстановка значений из модели данных в формате «`${}`», поэтому для значений типа `String` (и в какой-то мере для значений типа `Integer`) конструкция `env-name=«productId»` эквивалентна конструкции `value=«${productId}»`. Рассмотрим пример (рис. 5.16) — определение формы «SimpleForm» в файле `%OFBIZ_HOME%\specialized\bim_module\webapp\website\screens\forms.xml`:

```
<form name="SimpleForm" type="single" default-title-style="tableheadtext" default-widget-style="inputBox" default-tooltip-style="tabletext" target="test_scr" default-map-name="simplemap">
  <actions>
    <set field="simplemap.a" value="first default value"/>
    <set field="simplemap.b" value="second default value"/>
  </actions>
  <field name="a" title="First">
    <text/>
  </field>
  <field name="b" title="Second">
    <display/>
  </field>
  <field name="submitButton" title="Submit" widget-style="standardSubmit">
    <submit button-type="button"/>
  </field>
</form>
```

Рис. 5.16. Фрагмент XML-файла с определением формы

При подготовке формы сначала выполняется инициализация двух полей в ассоциативном массиве `simplemap`. В теге `<form>` (атрибут «`default-map-name`») указывается, что именно этот массив используется в качестве источника данных для полей формы.

Источник данных используется для инициализации и хранения значений двух полей с именами «*a*» и «*b*». Первое поле позволяет редактировать свое содержимое, а второе — только просматривать исходное значение. В форме определена кнопка «`SubmitButton`», при ее нажатии содержимое полей отправляется на сервер.

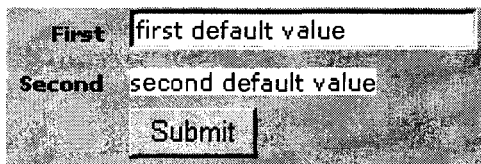
В определении формы, в атрибуте «`target`» указано имя запроса, которое передается в составе URI на сервер OFBIZ. По имени запроса будет определено, какое представление потребуется пользователю в результате обработки

переданных из формы данных. В нашем случае возвращается то же представление «test\_scr».

В определение экрана эта форма включена с помощью тега `include-form`:

```
<include-form name="SimpleForm"
location="component://bim_module/webapp/website/screens/forms.xml"/>
```

В результате пользователь видит на экране такой интерфейс, приведенный на рис. 5.17 (показан только фрагмент экрана с формой).



The image shows a portion of a web browser screen displaying a form. On the left, there are two labels: "First" and "Second". To the right of "First" is a text input field containing the text "first default value". To the right of "Second" is another text input field containing the text "second default value". Below these two input fields is a button labeled "Submit".

Рис. 5.17. Визуальное представление формы на экране Web-браузера



Рис. 5.17. Визуальное представление меню экране Web-браузера

### 5.3.2. Меню

Меню традиционно используется для быстрого вызова команд из заданного списка. В системе OFBIZ каждая команда меню обычно ассоциируется с именем определенного запроса — при выборе пункта меню происходит передача соответствующего URI для подготовки и отображения связанного с этим запросом представления на экране браузера (рис. 5.18).

Структура элементов, предназначенных для описания меню, представлена в виде UML-диаграммы на рис. 5.19. Этой диаграмме соответствует XML-схема

`%OFBIZ_HOME%\framework\widget\dtd\widget-menu.xsd`.



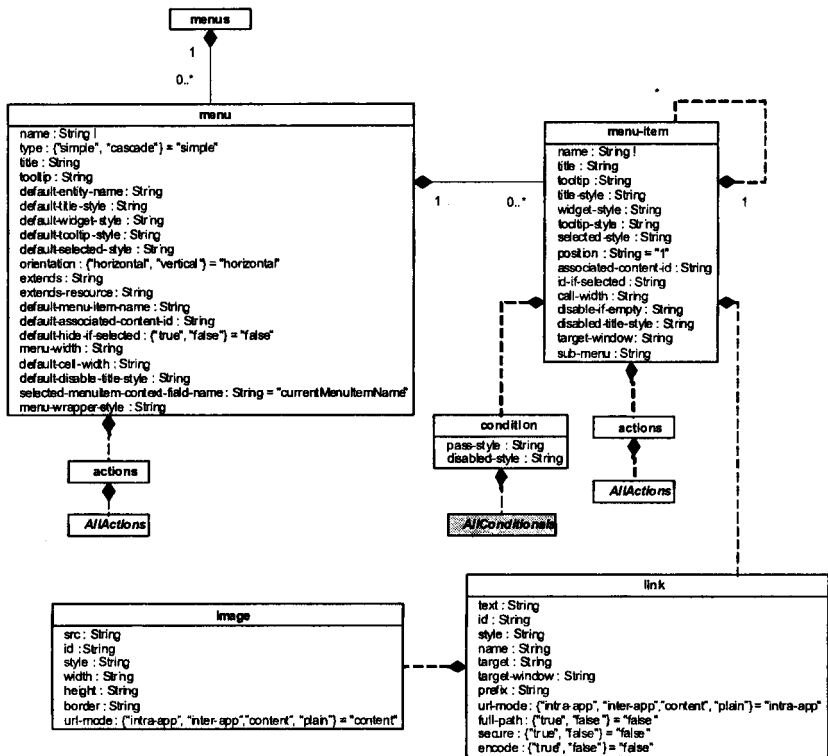


Рис. 5.19. Элементы для определения меню

Из диаграммы видно, что элемент верхнего уровня «menus» может содержать в себе несколько меню, различаемых по имени. Характеристики каждого меню задаются значениями разных атрибутов в теге <menu>.

Рассмотрим кратко важнейшие характеристики меню в системе OFBIZ. Например, предусмотрено два типа меню:

- ◆ simple — простое меню, где каждая команда не имеет подчиненных под-меню;
- ◆ cascade — каскадное меню, где некоторым командам соответствует определенное подчиненное меню со своим набором команд.

Задать нужный тип меню можно с помощью атрибута «type»<sup>24</sup>. При определении меню можно также выяснить его ориентацию путем задания значения атрибута «orientation». Он может принимать одно из двух значений: horizontal (команды располагаются в строку) или vertical (команды располагаются в ряд).

Для динамической инициализации различных параметров в определении меню в составе тега <menu> используется вложенный тег <actions>. В нем можно указывать такие же действия, что и в разделе «actions» в определении экрана.

Каждый отдельный пункт в составе меню определяется с помощью тега <menu-item>. Внешний вид различных состояний пункта меню задается значениями атрибутов: title-style, widget-style, selected-style, disabled-title-style. В случае реализации меню в виде HTML-страницы каждому имени стиля соответствует определение стиля из каскадной стилевой таблицы (см. информацию по каскадным стилевым таблицам в приложении).

Доступность (активность) каждого отдельного пункта можно определить в подчиненном теге <condition>. Внутри него можно использовать те же вложенные теги для проверки различных условий, что и в теге <condition> при описании экрана. Поэтому подробно наследники элемента AllConditionals здесь не рассматриваются.

При определении каждого отдельного пункта меню можно использовать тег <actions>, в котором перечисляются последовательно выполняемые действия (изменение модели данных, вызов служб и т.д.). На определение действий распространяются те же правила, что и при определении экрана, поэтому внутренняя структура тега <actions> здесь не раскрывается.

Реакция по выбору пункта меню определяется содержанием вложенного тега <link>. В атрибуте «target» этого тега указывается имя того представления, которое должно быть передано пользователю как ответ на выполнение команды меню. Определением атрибута «target-window» можно указать специфическое месторасположение представле-

---

<sup>24</sup> В текущей реализации поддерживаются только меню с типом «simple».

ния. В случае использования Web-интерфейса значения этого атрибута могут быть выбраны из следующего списка:

- ◆ «\_top» — результат обработки запроса (новое представление) открывается в самом внешнем окне;
- ◆ «\_self» — результат обработки запроса (новое представление) открывается в собственном окне;
- ◆ «\_parent» — результат обработки запроса (новое представление) открывается в родительском окне;
- ◆ «\_blank» — результат обработки запроса (новое представление) открывается в новом окне.

В качестве визуального оформления пункта меню может использоваться произвольное графическое изображение, параметры которого определяются во вложенном элементе «image». Правила именования изображения такие же, как и в языке HTML. Таким образом, в атрибуте «src» элемента «image» задается полный или частичный URI.

В качестве визуального оформления пункта меню может использоваться произвольное графическое изображение, параметры которого определяются во вложенном теге <image>. Правила именования изображения такие же, как и в языке HTML. Таким образом, в атрибуте «src» тега <image> задается полный или частичный URI.

Подробности по другим атрибутам тегов можно выяснить путем анализа содержимого XML-схемы или исходного кода, предназначенного для превращения XML-описания меню в правильно сформированный фрагмент HTML:

```
%OFBIZ_HOME%\framework\widget\src\org\ofbiz\widget\html\HtmlMenuRenderer.java.
```

### 5.3.3. Деревья

С назначением и возможностями этого элемента графического интерфейса вы впервые познакомились, изучая особенности работы с библиотекой Swing. Конечно, в языке OFBIZ-Widgets возможности дерева не такие богатые, как в библиотеке Swing, тем не менее, даже на экране Web-браузера с использованием стандартных возможностей языка HTML удастся достичь вполне привлекательного интерфейса (рис. 5.20).

В языке OFBIZ-Widgets интерфейсный элемент tree включается в структуру экрана с помощью тега <include-tree>. Его атрибуты аналогичны атрибутам тегов <include-form> и <include-menu>. Можно догадаться, что описание дерева обычно располагается в отдельном XML-файле, а правила по построению дерева формализованы в виде XML-схемы (эта схема располагается в файле

`%OFBIZ_HOME%\framework\widget\dtd\widget-tree.xsd`).

Наиболее значимые части XML-схемы приведены на рис. 5.21 в форме UML-диаграммы.

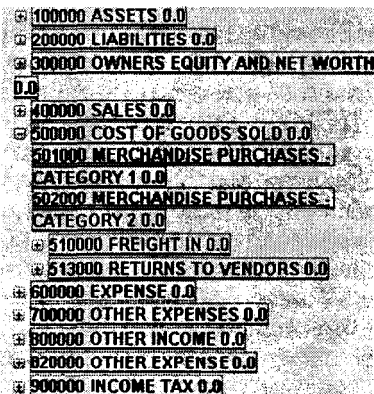


Рис. 5.20. Визуальное представление дерева на экране Web-браузера

Дерево состоит из узлов разных типов. В определении каждого типа узла указывается имя (атрибут «name»). Для визуального отображения могут использоваться какой-то экран, текстовая метка или ссылка на другое представление.

Перечисление типов узлов производится последовательно. Связь родительского и подчиненного узла в дереве определяется вложенным тегом <sub-node>, указываемым при определении типа узла. Связи могут быть рекурсивными, т.е. в качестве аргумента sub-node можно указывать имя того типа узла, в определение которого вложен сам тег <sub-node>.

Построение реального экземпляра дерева по описанию происходит с использованием возможностей ЦУЭД. В теге

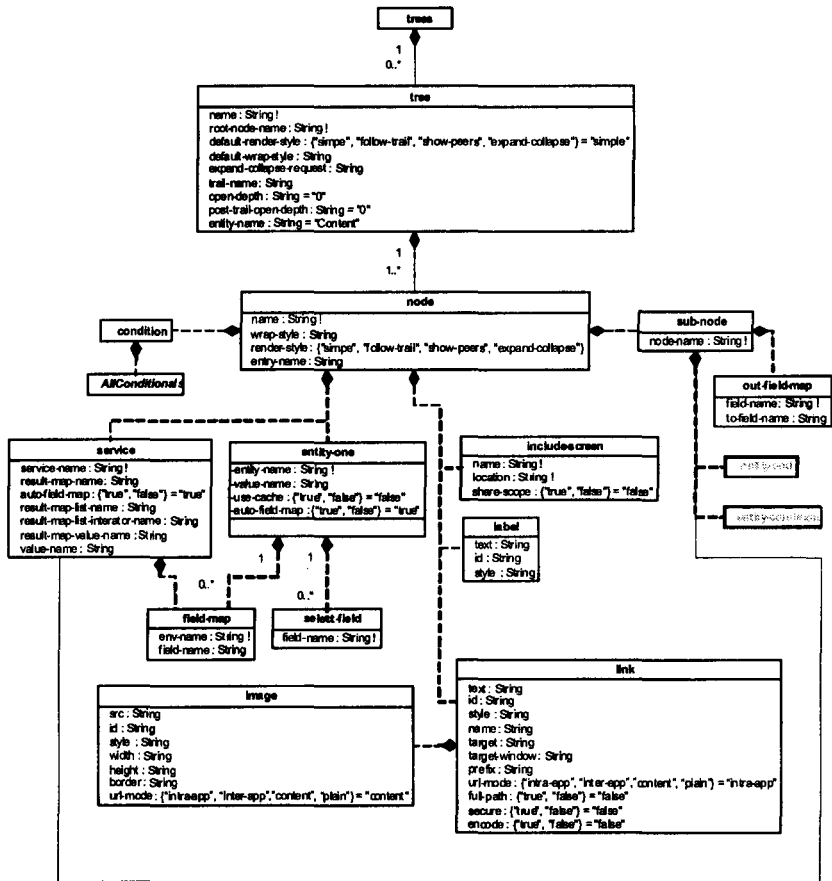


Рис. 5.21. Элементы для определения дерева

<sub-node> указывается тип элемента данных и отображение полей элемента данных на поля модели данных. Сколько из ЦУЭ выдается элементов данных, столько подчиненных узлов и создается. При этом каждому узлу соответствует свой экземпляр элемента данных. Имя элемента данных и условия по выборке из ЦУЭ определяются во вложенном теге <entity-and>.

Указание соответствия между полями элемента данных и полями модели данных задается с помощью вложенного тега `<field-map>`.

След (trail) определяет состояние дерева (открытые и закрытые вершины). След передается в качестве одного из параметров запроса в контроллер, поэтому в разных представлениях можно воспроизвести текущую ситуацию в дереве, используя поле модели данных `currentNodeTrailPiped`.

К сожалению, интерфейсный элемент дерево в системе OFBIZ находится в стадии разработки, поэтому пока не обеспечено его устойчивое поведение и многие возможности не документированы. Поэтому в дальнейших примерах мы его использовать не будем.

### 5.3.4. Использование шаблона проектирования «Декоратор»

Для того чтобы повысить повторную используемость созданных определений экранов, форм, меню и деревьев при разработке интерфейсов различных приложений, разработчики системы OFBIZ рекомендуют следовать общим принципам шаблона проектирования «декоратор». В этом случае проектировщик должен в начале проанализировать структуру интерфейсов всех запланированных в компоненте приложений и выделить общие для них части. Например, в большинстве приложений в состав всех интерфейсов будут входить единые заголовки и «подвалы» экранов, стандартные меню, формы поиска и т.д.<sup>25</sup> Такие общие элементы группируются в составе определения особого экрана, носящего в системе OFBIZ название «экран-декоратор» (decorator-screen).

«Экран-декоратор» определяется с помощью уже знакомого тега `<screen>`, причем для определения условий, действий и его внутренней структуры можно использовать все известные элементы, собранные в XML-схеме `widget-screen.xsd`. Существует лишь одно важное дополнение — в составе определения внешнего вида «экрана-декоратора»

---

<sup>25</sup> Неизменяемыми могут оставаться лишь общая структура и взаимное расположение различных интерфейсных элементов. Конкретное содержимое вполне может быть изменяемым.

(внутри тега `<widgets>`) можно использовать новый тег `<decorator-section-include>`. Он определяет модифицируемую часть экрана — «гнездо», куда различные специфичные интерфейсы могут «вкладывать» свое особое содержимое. Тег `<decorator-section-include>` имеет один обязательный атрибут — «name». В составе одного «экрана-декоратора» может находиться произвольное число тегов `<decorator-section-include>` с различными именами.

При наличии «экрана-декоратора», определяющего общую часть различных интерфейсов, их проектирование заметно облегчается. Создавая новый специфичный интерфейс (экран, в терминах OFBIZ), разработчик должен решить, какой «экран-декоратор» следует взять за основу и в какие «гнезда» необходимо «вложить» специфическое содержимое. В секции «widgets» разрабатываемого экрана нужно определить структуру этого специфического содержимого, а с использованием тегов `<decorator-screen>` и `<decorator-section>` указать, какой «экран-декоратор» нужно использовать и какое «гнездо» наполнять.

Рассмотрим, как на основе шаблона «декоратор» реализуется единый стиль стандартного графического интерфейса в системе OFBIZ. Пусть для группы экранов определена одна и та же общая структура, представленная на рис. 5.22. Во всех интерфейсах используется общий заголовок

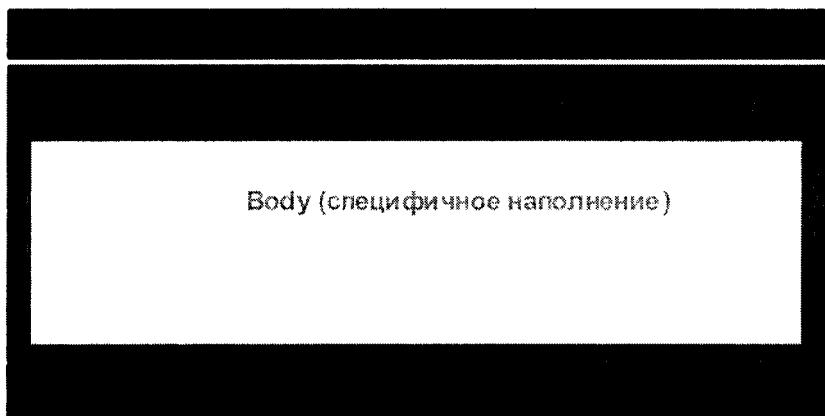


Рис. 5.22. Единая структура всех экранов

(Header) и «подвал» (Footer), а специфический для каждого приложения интерфейс размещается в «гнезде» «Body». Поэтому в определении различных интерфейсов можно использовать один и тот же «экран-декоратор».

В XML-файле с определениями экранов «экран-декоратор» задается так (рис. 5.23):

```
<!-- DECORATOR SCREEN -->
<screen name="main-decorator">
  <section>
    <actions>
      <!-- base/top/specific map first, then more common map added for shared labels
-->
      <property-map resource="BimUiLabels" map-name="uiLabelMap"
global="true"/>
      <property-map resource="CommonUiLabels" map-name="uiLabelMap"
global="true"/>
      <set field="layoutSettings.companyName" from-
field="uiLabelMap.BimCompanyName"
      <set field="person.firstName" value="Test Bim Person name " global="true"/>
      <set field="person.lastName" value="LN" global="true"/>
      <set field="BIMLogin" from-field="uiLabelMap.BIMLogin"/>
    </actions>
    <widgets>
      <!-- render header -->
      <platform-specific>
        <html>
          <html-template
location="component://bim_module/webapp/website/includes/common/header.ftl"/>
        </html>
      </platform-specific>
      <!-- Application-specific Content Area -->
      <container>
        <decorator-section-include name="Body"/>
      </container>
    </container>
      <!-- render footer -->
      <platform-specific>
        <html>
          <html-template
location="component://bim_module/webapp/website/includes/common/footer.ftl"/>
        </html>
      </platform-specific>
    </widgets>
  </section>
</screen>
```

Рис. 5.25. Определение «экрана-декоратора»



Пусть теперь в приложении нужно определить два специфических интерфейса — экраны с именами «screen\_A» и «screen\_B». В этом же XML-файле, где ранее был определен «экран-декоратор», можно расположить определение специфичных экранов (рис. 5.24):

```
<!-- DECORATOR SCREEN -->
<screen name="main-decorator">
.....
</screen>

<!--APPLICATION-SPECIFIC SCREENS -->
<screen name="screen_A">
  <section>
    <widgets>
      <decorator-screen name="main-decorator">
        <decorator-section name="Body">
          <include-form name="SimpleFormA"
            location="component://bim_module/webapp/website/screens/forms.xml"/>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>

<screen name="screen_B">
  <section>
    <widgets>
      <decorator-screen name="main-decorator">
        <decorator-section name="Body">
          <include-form name="SimpleFormB"
            location="component://bim_module/webapp/website/screens/forms.xml"/>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
```

**Рис. 5.24.** Определение специфичных частей

Из рис. 5.24 видно, что определение специфичных интерфейсов, основанных на общем «экране-декораторе», довольно компактно. Кроме того, изменив лишь в одном месте определение какой-то части «экрана-декоратора», мы сразу же сделаем изменения доступными во всех специфичных интерфейсах.

## 5.4. Методы изучения функциональности сложных программных систем

Система OFBIZ является удобным примером, на котором можно продемонстрировать общие принципы изучения методов реализации функциональности сложного программного продукта по исходному коду и примерам. В том случае, когда необходимо решить определенную задачу с помощью мало знакомой программной системы, рекомендуется последовательно выполнить следующие шаги.

1. Точно определить, какую функциональность вы хотите реализовать в собственном приложении.

2. Рассмотреть имеющиеся примеры интерфейсов в стандартных приложениях и выбрать наиболее близкие к вашей задаче аналоги.

3. Определить, в каких XML-файлах конфигурации и параметров описаны заинтересовавшие вас примеры, и внимательно изучить их, а потом использовать в качестве шаблонов.

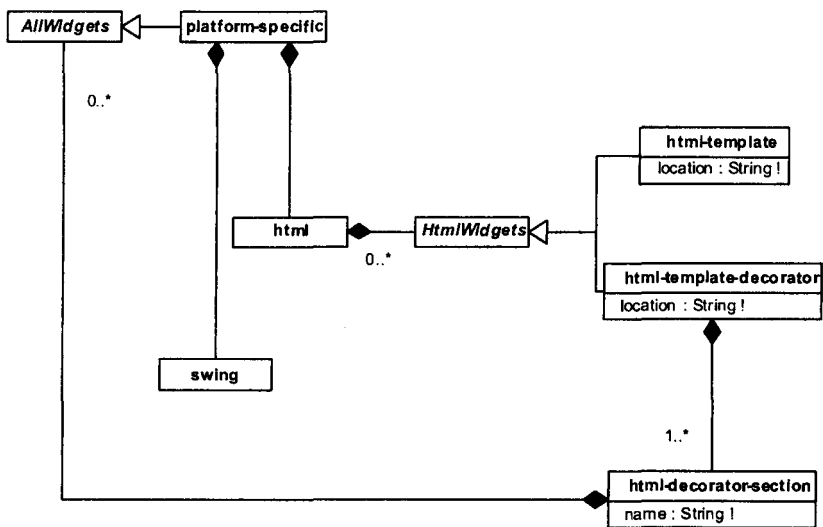


Рис. 5.25. Элементы языка OFBIZ-Widgets для определения платформно зависимых фрагментов интерфейса

4. Если назначение определенных параметров или XML-тегов неясно, найти файлы исходного кода, в которых происходит разбор этих параметров или тегов, и попытаться понять алгоритм.

5. Если алгоритмы слишком сложны, то обратиться на сайт разработчиков. Там внимательно изучить ответы и вопросы, собранные в списках почтовой рассылки (Mailing Lists) или в системе интерактивной помощи (на сайте OFBIZ для этой цели служит раздел Wiki).

К сожалению, в настоящее время выразительные средства языка OFBIZ-Widgets еще не позволяют полностью освободить проектировщика от создания платформно зависимых фрагментов интерфейса. Поэтому значительную долю описания интерфейса, предназначенного для использования в Web-приложении, составляют фрагменты на языке HTML (рис. 5.25).

Назначение основных элементов, изображенных на рис. 5.25, дано в табл. 5.4.

Однако система OFBIZ предоставляет возможность не только включать статические фрагменты, но и проводить генерацию HTML-выражений «на лету», подставляя значения переменных, вызывая службы, используя условную вставку и другие возможности. Модульная и расширяемая архитектура позволяет для этой цели использовать различные технологии, но наиболее популярной является система FreeMarker.

*Таблица 5.4*

**Назначение основных элементов**

<b>Название</b>	<b>Назначение</b>
platform-specific	Определяет фрагмент интерфейса, описание которого зависит от какой-либо платформы или технологии
html	Указывает на то, что в описании фрагмента используется специфика языка HTML
swing	Указывает на то, что в описании фрагмента используется специфика библиотеки Swing
html-template	Определяет внешний файл, содержащий описание фрагмента на языке HTML

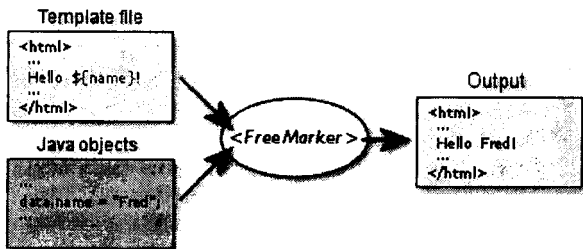


Рис. 5.26. Принцип работы системы FreeMarker

Система FreeMarker является независимо разрабатываемой одним из авторов OFBIZ библиотекой Java-классов. Она реализует богатые возможности по генерации произвольных текстов (начиная от HTML и заканчивая исходным кодом) на основе шаблонов.

Документацию, примеры и исходный код системы FreeMarker можно получить бесплатно с сайта разработчиков <http://www.freemarker.org>. По замыслу разработчиков система FreeMarker воплощает идею шаблона проектирования MVC, позволяя работать совместно различным категориям специалистов: дизайнеры разрабатывают шаблоны (общий стиль, графика и другие аспекты визуального представления), программисты проектируют и реализуют различные Java-классы, преобразующие и подготавливающие данные для отображения (рис. 5.26). На долю системы FreeMarker остается автоматическая генерация законченных текстовых страниц, в которых содержатся подготовленные данные в визуальном формате, определенном шаблонами.

Для разработки шаблонов в системе FreeMarker используется расширение языка HTML. Кроме стандартных тегов языка HTML это расширение позволяет использовать при разработке шаблонов:

- ◆ все стандартные процедурные конструкции: include, if/elseif/else, loop;
- ◆ создание и модификацию переменных;
- ◆ сложные выражения для вычисления значений с использованием:

- операций со строками (конкатенация, выделение под-строк, изменение регистра букв и т.п.),
- арифметических операций с указанной точностью,
- логических операций,
- порядковых и ассоциативных массивов,
- пользовательских процедур и функций;
- ◆ макроопределения (в том числе и вложенные макроопределения) с доступом к параметрам по имени или позиции, например:

```
<@myMacro color="red" width=2>...</@myMacro>;
```

- ◆ поддержку различных пространств имен (Name-spaces) для создания и поддержки повторно-используемых библиотек макроопределений или декомпозиции больших проектов на отдельные модули;
- ◆ сложные правила трансформации выходных данных (кодирование и сжатие текста, выделение текста и т.п.) в результате применения нескольких вложенных друг в друга шаблонов, определенных разработчиком.

Для подготовки данных и вставки их в шаблон в системе FreeMarker используется такой же механизм модели данных, как и в языке OFBIZ-Widget. Однако кроме ассоциативных массивов, скаляров и последовательностей в составе модели данных FreeMarker можно использовать несколько иных типов данных.

Приведем небольшой пример, иллюстрирующий процесс создания законченной HTML-страницы на основе шаблона и модели данных.

Шаблон выглядит почти так же, как и обычная HTML-страница, за исключением включения инструкций, определяющих что и куда добавлять к шаблону во время генерации. В качестве такой инструкции может использоваться уже знакомое вам выражение `${<нуть_к_переменной>}` (рис. 5.27).

Значения для полей `latestProduct.url` и `latestProduct.name` FreeMarker выбирает во время генерации из своей модели данных. Например, в качестве модели может использоваться дерево, показанное на рис. 5.28.

Подставляя в указанные места шаблона актуальную информацию из модели, FreeMarker получает в результате готовую HTML-страницу (рис. 5.29):

```

<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome #{user}!</h1>
  <p>Our latest product
  <a href="#{latestProduct.url}">#{latestProduct.name}</a>!
</body>
</html>

```

Рис. 5.27. Пример FTL-шаблона

```

(root)
|
+- user = "Big Joe"
|
+- latestProduct
  |
  +- url = "products/greenmouse.html"
  |
  +- name = "green mouse"

```

Рис. 5.28. Используемая в примере модель данных

```

<html>
<head>
  <title>Welcome!</title>
</head>
<body>
  <h1>Welcome Big Joe!</h1>
  <p>Our latest product:
  <a href="products/greenmouse.html">green mouse</a>!
</body>
</html>

```

Рис. 5.29. Результат генерации

Теперь рассмотрим иные возможности, используемые при создании шаблонов системы FreeMarker. Вырожденным случаем шаблона является обычная статичная HTML-страница, которая передается клиенту в том виде, в каком она хранится на сервере. А в общем случае разработчик шаблонов в системе FreeMarker может включать в текст HTML-страницы расширения трех видов:

1) `{...}` — подстановка значения поля из модели данных или результата выражения;

2) специальные FTL-теги (FTL — от FreeMarker Template Language), которые определяют определенную инструкцию для выполнения системой FreeMarker. После выполнения инструкции ее результат заместит соответствующий FTL-тег. Существуют две разновидности FTL-тегов: встроенные и определенные пользователями. Встроенные FTL-теги начинаются с подстроки «<#», а определенные пользователем — с «<@». Так же, как и в языках HTML и XML, возможно несколько вариантов окончания тегов:

◆ Start-tag: `<#directivename parameters><@directivename parameters>`;

◆ End-tag: `</#directivename></@directivename>`;

◆ Empty directive tag: `<#directivename parameters/><@directivename parameters/>`.

3) комментарии системы FreeMarker в виде `<#—` и `—>`. Любой текст, заключенный между этими комментариями, в готовую страницу не попадет.

При использовании системы FreeMarker нужно помнить, что она чувствительна к регистру, поэтому, например, `{name}`, `{Name}` и `{NAME}` обозначают три разных поля-переменных.

Перечислим наиболее важные встроенные FTL-теги, использующиеся при разработке шаблонов.

◆ *Инструкция if*

Использование этой инструкции позволяет исключить из результата определенные фрагменты шаблона на основании проверки условия.

◆ *Инструкция list*

Эта инструкция полезна в том случае, когда требуется в результате создать перечисление.

◆ *Инструкция include*

Эта инструкция позволяет включить содержимое произвольного файла в текущий шаблон.

Различные директивы системы FreeMarker можно многократно повторять, комбинировать и вкладывать друг в друга, в зависимости от потребности.

◆ **Инструкция *default***

Эта инструкция позволяет переопределять переменные по умолчанию.

◆ **Инструкция *if\_exists***

Эта инструкция вычисляет значение уже определенной переменной.

## ***5.5. Особенности интеграции с системой OFBIZ***

На протяжении длительного срока существования системы OFBIZ Free-Marker являлся основным средством для динамичной генерации HTML-страниц. Поэтому многие части OFBIZ и FreeMarker тесно интегрированы друг с другом. Основным, что требуется помнить, работая с шаблонами и моделью данных FreeMarker в составе системы OFBIZ, является кеширование в оперативной памяти FTL-шаблонов и доступные по умолчанию поля модели данных с предопределенными именами, добавляемые автоматически системой OFBIZ.

При первом запросе на генерацию страницы по FTL-шаблону система OFBIZ помещает этот шаблон в один из своих внутренних буферов в оперативной памяти (в так называемый кеш). Это приводит к тому, что последующее изменение содержимого шаблона в файле на диске не вызывает изменений на экране пользователя. Для того, чтобы во время активной разработки интерфейса не приходилось перезапускать OFBIZ каждый раз, когда требуется актуализировать сделанные в шаблоне изменения, рекомендуется установить время старения данных в кеше шаблонов в значение 10 000 (100 сек). Время нахождения в кеше системы OFBIZ задается в файле параметров %OFBIZ\_HOME%\ base\config\cache.properties. В данном случае нас должны интересовать параметры для двух кешей:

- ◆ `template.ftl.general.expireTime=10000;`
- ◆ `template.ftl.location.expireTime=10000.`



Естественно, что изменения времени нахождения в кеше вступят в силу лишь после перезапуска системы OFBIZ. Можно также порекомендовать удобное средство для интерактивного управления кешами системы OFBIZ в обычном Web-браузере. В Интернет-приложении «webtools» для этого используется ссылка «Cache maintenance». В том случае, если система OFBIZ запущена на локальной машине, то соответствующий URL выглядит так:

<https://127.0.0.1:8443/webtools/control/FindUtilCache>.

При разработке FTL-шаблонов в составе системы OFBIZ разработчику доступны все поля модели данных OFBIZ. Кроме того, модель данных содержит переменную Static, которая является ассоциативным массивом и обеспечивает доступ к статическим методам и полям произвольного класса в рамках системы OFBIZ (например, `Static["java.lang.System"].currentTimeMillis()`).

При использовании ассоциативных массивов нередко требуется получить список имен полей, хранящихся в нем. В системе FreeMarker ассоциативные массивы обладают служебными методами keys и values, которые возвращают последовательности имен и значений полей. Использование этой возможности проиллюстрировано на следующем примере FTL-шаблона:

```
<#assign keys = h?keys>
<#list keys as key>${key} = ${h[key]}; </#list>
```

При разработке шаблонов также можно использовать группу predefined макросов и функций:

Макрос/функция	Назначение
1	2
ofbizUrl	содержит полный сетевой путь (URL) к сервлету-контроллеру данного Web-приложения
ofbizContentUrl	содержит полный сетевой путь (URL) к подготавливаемой странице. Например, <code>&lt;@ofbizContentUrl&gt;/images/selectall.js&lt;/@ofbizContentUrl&gt;</code>
ofbizCurrency	содержит локализованное обозначение символа валюты. Например, <code>&lt;@ofbizCurrency amount="32.50" isoCode="UTF-8" locale="fr_FR"/&gt;</code>

1	2
setRequestAttribute	функция изменяет значение определенного элемента данных, переданных из формы. Например, <code>#{setRequestAttribute("listIndex", listIndex)}</code>
barcode	
editRenderSubContent	
renderSubContent	вставляет определенные данные из системы управления содержанием OFBIZ. Например, <code>&lt;@renderSubContentCache subContentId="ECMC180000"/&gt;</code>
loopSubContent	
traverseSubContent	
renderWrappedText	
editRenderSubContent Cache	
renderSubContent Cache	
loopSubContentCache	
traverseSubContent Cache	
wrapSubContent Cache	
checkPermission	Вставляет содержимое, если текущие параметры безопасности соответствуют определенным ограничениям. Например, <code>&lt;@checkPermission mode="equals" entityOperation="CREATE" targetOperation="HAS_USER_ROLE" &gt; &lt;a class="tabButton" href="&lt;@ofbizUrl&gt;createforumresponse?contentIdTo=\${requestParameters.contentId}&amp;nodeTrailCsv=\${nodeTrailCsv?if_exists} &lt;/@ofbizUrl&gt;" &gt;Respond&lt;/a&gt; &lt;/@checkPermission&gt;</code>
injectNodeTrailCsv	Вставляет определенное содержимое по переданному из формы следу (trail) меню OFBIZ. Например, <code>&lt;@injectNodeTrailCsv subContentId=content.contentId redo="true" contentAssocTypeId="PUBLISH_LINK"&gt;</code>

1	2
menuWrap	
limitedSubContent	
renderSubContent AsText	
renderContentAsText	

## 5.6. Готовый пример с *ftl*

Рассмотрим пример описания общего заголовка экрана, в котором используются возможности FreeMarker. В этом примере активно применяются расширенные возможности языка HTML под названием «каскадные таблицы стилей» (Cascading Style Sheets). Если вы с ними раньше не встречались, то рекомендуем перед рассмотрением примера изучить основы CSS в приложении.

Текст шаблона представлен на рис. 5.30. В нем выделены те фрагменты, которые относятся к языку FTL. Обычным шрифтом набраны выражения на языке HTML.

```

1. <!doctype HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
2. <!-- Copyright (c) 2001-2004 The Open For Business Project - www.ofbiz.org -->
3. <!--
4. * Copyright (c) 2001-2004 The Open For Business Project - www.ofbiz.org
5. * @author Andy Zeneski (jaz@ofbiz.org)
6. * @author Olivier Heintz (olivier.heintz@nereide.biz)
7. * @version $Rev: 4140 $
8. * @since 2.1
9. -->
10. <#if (requestAttributes.uiLabelMap)?exists>
11.     <#assign uiLabelMap = requestAttributes.uiLabelMap>
12. </#if>
13. <#if (requestAttributes.layoutSettings)?exists>
14.     <#assign layoutSettings = requestAttributes.layoutSettings>
15. </#if>
16. <#if (requestAttributes.locale)?exists>
17.     <#assign locale = requestAttributes.locale>
18. </#if>
19. <#if (requestAttributes.availableLocales)?exists>
20.     <#assign availableLocales = requestAttributes.availableLocales>
21. ....

```

Рис. 5.30. Пример шаблона, используемого в системе OFBIZ

```

21. <#if>
22. <#if (requestAttributes.person)?exists>
23.     <#assign person = requestAttributes.person>
24. </#if>
25. <#if (requestAttributes.partyGroup)?exists>
26.     <#assign partyGroup = requestAttributes.partyGroup>
27. </#if>
28. <html>
29. <head>
30. <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
31. <title>${layoutSettings.companyName}</title>
32. <#if (page.titleProperty)?has_content>
33.     ${uiLabelMap[page.titleProperty]}
34. <#else>
35.     ${page.title}?if_exists
36. </#if>
37. </title>
38. <script language="javascript"
src='<@ofbizContentUri>/images/calendar1.js</@ofbizContentUri>'
type="text/javascript"></script>
39. <script language="javascript"
src='<@ofbizContentUri>/images/selectall.js</@ofbizContentUri>' type="text/javascript">
40. <link rel="stylesheet" href='<@ofbizContentUri>/images/bim_maincss.css</@ofbizContentUri>'
type="text/css">
41. </head>
42.
43. <body>
44. <table border=0 width="100%" cellspacing="0" cellpadding="0" class="headerboxoutside">
45. <tr>
46. <td width="100%">
47. <table width="100%" border="0" cellspacing="0" cellpadding="0" class="headerboxtop">
48. <tr>
49. <#if layoutSettings.headerImageUrl?exists>
50. <td align="left" width="1%">
51. <img alt="${layoutSettings.companyName}"
src='<@ofbizContentUri>${layoutSettings.headerImageUrl}</@ofbizContentUri>'
52. </td>
53. <#if>
54. <td align="right" width="1%" nowrap
55. <#if layoutSettings.headerRightBackgroundUrl?has_content>
background='${layoutSettings.headerRightBackgroundUrl}' >
56. <#if>
57. <#if person?has_content>
58. <div class="insideHeaderText">
59.
${uiLabelMap.CommonWelcome}&nbsp;${person.firstName?if_exists}&nbsp;${person.lastName?if_exists}!
60. </div>
61. <#elseif partyGroup?has_content>
62. <div
class="insideHeaderText">${uiLabelMap.CommonWelcome}&nbsp;${partyGroup.groupName?if_exists}!</div>
63. <#else>
64. <div class="insideHeaderText">${uiLabelMap.CommonWelcome}!</div>
65. <#if>

```

Рис. 5.30. Пример шаблона, используемого в системе OFBIZ  
(продолжение)

```

66.     <div
class="insideHeaderText">&nbsp;${Static["org.ofbiz.base.util.UtilDateTime"].nowTimestamp().t
oString()}</div>
67.     <div class="insideHeaderText">
68.         <form method="POST" action="@@ofbizUrl/setSessionLocale</ofbizUrl">
style="margin: 0;">
69.             <select name="locale" class="selectBox">
70.                 <option value="${locale.toString()}">${locale.getDisplayName(locale)}</option>
71.                 <option value="${locale.toString()}>----</option>
72.                 <#list availableLocales as availableLocale>
73.                     <option
value="${availableLocale.toString()}">${availableLocale.getDisplayName(locale)}</option>
74.                 </#list>
75.             </select>
76.             <input type="submit" value="{uiLabelMap.CommonSet}" class="smallSubmit"/>
77.         </form>
78.     </div>
79. </td>
80. </tr>
81. </table>
82. </td>
83. </tr>
84. </table>

```

**Рис. 5.30.** Пример шаблона, используемого в системе OFBIZ (окончание)

В этом шаблоне динамично создаются имя компании, параметры текущего пользователя, список из известных в системе национальных установок (локализаций). В строчках 10—27 из ассоциативного массива извлекаются необходимые для дальнейшей работы данные. В строчках 31—36 определяется содержимое заголовка HTML-страницы на основании содержимого поля с именем «titleProperty» из модели данных. На строчке 38 используется конструкция <@, чтобы динамично подставлять значение адреса сервера OFBIZ в начальную часть URI, определяющего расположение календаря. Та же конструкция применяется на строках 39 и 40. В строчках 49—53 используется тег условного включения содержимого <#if для того, чтобы вставлять динамически имя компании. Строчки 57—65 содержат фрагмент, в котором отображается имя текущего пользователя или группы. В строчке 66 стоит вызов статического метода nowTimeStamp класса org.ofbiz.base.util.UtilDateTime, который возвращает текущее время. Со строчки 68 по 76 динамично генерируется содержимое списка доступных локализаций.

## 5.7. Контрольные задания к гл. 5

Все задания основаны на расширении тестового web-приложения `bim_module`. Простейший вариант этого приложения находится в каталоге `%OFBIZ_HOME%\hot-deploy`. В первоначальном варианте в этом приложении определено одно представление `main`, реализованное на языке `OFBIZ-Widget`. Обратиться к нему можно после запуска системы `OFBIZ` на локальном сервере по URL

[http://localhost:8080/bim\\_module/main](http://localhost:8080/bim_module/main).

1. Создать в компоненте `bim_module` новое представление, передающееся пользователю как результат обработки запроса с именем, соответствующим фамилии студента (для этого не забудьте отредактировать файл

`bim_module/webapp/website/WEB-INF/controller.xml`).

Представление реализовать в виде экрана на языке `OFBIZ-Widgets` с включением `FTL`-шаблонов. Нужно придерживаться структуры экрана, определенной на рис. 5.22. Содержимое основной рабочей области — произвольно (например, приветственный текст, цитата и т.п.). Определения собственных экранов можно размещать в уже существующем файле

`bim_module/webapp/website/screens/MainScreens.xml`.

2. Расширить созданное представление, добавив в представление форму с полями: `Salutation`, `UserName` и кнопкой. После заполнения формы должен посылаться запрос на выдачу того же представления. При подготовке представления данные, введенные пользователем в форму, должны отображаться в виде текста в основной области экрана. Если данные отсутствуют (при первом заходе), то должно выводиться особое текстовое значение «N\A».

3. Разработать для созданного представления оригинальную цветовую и шрифтовую стилистику, создав собственный `CSS`-файл и указав его в `FTL`-шаблоне.

4. Добавить в представление меню, содержащее три команды. При этом в файле `controller.xml` нужно будет добавить три новых запроса и три произвольных представления (можно использовать тип `http` для выдачи пользователю страниц с других сайтов).

5. Прочитайте в приложении раздел о локализации интерфейса в системе `OFBIZ`. Модифицируйте результат

задания № 2 так, чтобы пользователь мог выбирать нужную ему локализацию и в зависимости от выбора — имена полей — формы и другая информация отображалась бы на выбранном языке. Используйте в работе имеющийся в компоненте `bim_module` FTL-шаблон для общего заголовка экрана.

6. Прочитайте в приложении раздел о стандартных параметрах HTTP-запроса, модифицируйте результат задания № 1 так, чтобы на экран пользователю выводилось как можно больше служебной информации (как минимум — тип пользовательского Web-браузера).

## ***5.8. Приложения к главе 5***

### ***5.8.1. Вопросы локализации интерфейса***

Рано или поздно при разработке графического интерфейса возникает вопрос об интернационализации программы — возможности отображения информации на предпочтительном для пользователя языке, с учетом национальных особенностей в формате дат, названиях денежных единиц и т.п. Причем наилучшим решением этой задачи является предоставление самому пользователю возможности выбора желаемой конфигурации из списка известных системе. Но не все так просто, даже для одного языка — английского. Он является официальным языком не только в США, но и в Англии или Австралии. И во всех этих странах используют различные его варианты. Кроме того, эти страны по-разному форматируют дату, время, числа, денежные единицы и т.п.

Для того чтобы унифицировать принципы учета различных национальных особенностей, при разработке программ было предложено использовать понятие «локаль» (`locale`) — точное определение культурных, политических особенностей отображения информации и языка определенного региона. Таким образом, локализация приложения — это его модификация для поддержки отображения информации в виде, характерном для какого-то конкретного региона (для определенной локализации). Рассмотрим, что предлагает разработчику в этом плане система OFBIZ.

Возможности локализации в системе OFBIZ основаны на стандартных средствах поддержки национальных особенностей языка Java. В этом языке используются стандартные обозначения для всех известных видов локалей (ISO-стандарт). Обозначение локали состоит из двух частей — начала названия официального языка и начала названия страны. Части разделяются между собой тире или дефисом. При указании локали разрешается не использовать вторую часть и обозначать лишь язык:

- ◆ en — английский язык без местных особенностей в форматах дат, валюты и пр.;

- ◆ en\_US — английский язык с учетом специфики дат, валюты и пр. США;

- ◆ fr — французский язык;

- ◆ ru — русский язык без местных особенностей в форматах дат, валюты и пр.;

- ◆ ru\_RU — русский язык с учетом специфики дат, валюты и пр. России;

- ◆ de — немецкий язык.

Система OFBIZ может сама определить, какой вариант файла с Java-свойствами нужно использовать в зависимости от установленной локали, например, в элементе «property-map». Для этого разработчик должен лишь придерживаться определенных правил по именованию файлов со свойствами, содержащих локализованные значения одних и тех же свойств. Эти правила просты:

- ◆ все файлы свойств должны находиться в каталоге config компонента;

- ◆ имена файлов, содержащих локализованные значения одних и тех же свойств, должны быть построены по такому шаблону:

`<CommonPropertyName>_<OfficialLocaleName>.properties,`

где `<CommonPropertyName>` — общее имя файла свойств, используемое, например, в элементе «property-map», а `<OfficialLocaleName>` — официальное обозначение локали.

Таким образом, любые текстовые и числовые константы, значения которых зависят от установленной локали,



могут быть вынесены в соответствующие файлы свойств, а во время работы программы — автоматически загружены с учетом пользовательских установок. Например, имена полей для форм могут быть помещены для каждого используемого языка в отдельный файл:

- ◆ config/labelsNames\_ru\_RU.properties;
- ◆ config/labelsNames\_en.properties;
- ◆ config/labelsNames\_fr.properties.

При создании содержимого файла ресурсов для русской локали нужно быть готовым к трудностям. В этом файле нельзя в качестве значений ресурсов использовать обычные русские буквы! Требуется вместо этого применять числовые коды русских букв в кодировке Unicode. Например, вместо того, чтобы написать в файле ресурсов:

```
helloWorld=Ѓёèââò, ìèð!!!,
```

нужно создать такую строчку

```
helloWorld=\u041fu0440\u0438\u0432\u0435\u0442,\n           \u043c\u0438\u0440!!!
```

К счастью, разработчики Java позаботились о пользователях и в составе стандартного JDK поставляют служебную программу `native2ascii`, выполняющую преобразование содержимого файлов в пользовательской кодировке в текст, закодированный на Unicode. Благодаря этому можно создать файл с русскоязычными ресурсами в любом текстовом редакторе привычным образом, набирая обычный текст, а затем провести конвертацию этого файла с помощью утилиты `native2ascii`.

Параметры вызова программы `native2ascii`.

```
native2ascii [options] [inputfile [outputfile]]
```

*Options:*

- ◆ `reverse`

Выполняет обратную перекодировку из Unicode в указанную кодировку.

- ◆ `encoding encoding_name`

Указывает имя кодировки, в которой набрано содержимое файла.

## 5.8.2. Информация по каскадным стилевым таблицам

Основным понятием CSS (Cascading Style Sheets) является «стиль» — т.е. набор правил оформления и форматирования, который может быть применен к различным элементам страницы. В стандартном HTML для присвоения какому-либо элементу определенных свойств (таких, как цвет, размер, положение на странице и т.п.) приходилось каждый раз описывать эти свойства, даже если на одной страничке должны располагаться 10 или 110 таких элементов, ничуть не отличающихся один от другого. Вы должны были десять или сто десять раз вставить один и тот же фрагмент HTML-кода в страничку, увеличивая размер файла и время загрузки на компьютер просматривающего ее пользователя.

CSS действует более удобным и экономичным способом. Для присвоения какому-либо элементу определенных характеристик вы должны один раз описать этот элемент и определить это описание как стиль, а в дальнейшем просто указывать, что элемент HTML-страницы, который вы хотите оформить соответствующим образом, должен принять свойства стиля, описанного вами.

Более того, вы можете сохранить описание стиля не в тексте вашей странички, а в отдельном файле — это позволит использовать описание стиля на любом количестве Web-страниц, а также изменить оформление любого количества страниц, исправив лишь описание стиля в одном (отдельном) файле. Кроме того, CSS позволяет работать со шрифтовым оформлением страниц на гораздо более высоком уровне, чем стандартный HTML, избегая излишнего утяжеления страниц графикой.

Существует четыре способа связывания документа и таблицы стилей:

- 1) внедрение — позволяет задавать все правила таблицы стилей непосредственно в самом документе;
- 2) встраивание в теги документа — позволяет изменять форматирование конкретных элементов страницы;
- 3) импортирование — позволяет встраивать в документ таблицу стилей, расположенную на сервере;

4) связывание — позволяет использовать одну таблицу стилей для форматирования многих страниц HTML.

### 5.8.3. Внедрение

Это вариант, при котором описание стилей располагается в коде Web-странички, внутри тега HEAD, в теге `<STYLE type="text/css">... </STYLE>`. В этом случае вы можете использовать эти стили для элементов, располагающихся в пределах одной HTML-странички. Параметр `type="text/css"` является обязательным и служит для указания браузеру использовать CSS.

Общий синтаксис при этом способе таков:

```
<HEAD>
<STYLE TYPE="text/css">
элемент {свойство: значение; свойство: значение}
</STYLE>
</HEAD>
```

Пример:

```
<HEAD>
<STYLE TYPE="text/css">
B {color: red; font-size: 120%}
</STYLE>
</HEAD>
<BODY>
...
<B>
Этот текст будет отображен в соответствии с описанием в заголовке:
красным цветом и размером шрифта в 120% от размера, принятого по
умолчанию. </B><BR>
<B STYLE="color:blue"> А этот текст будет отображаться синим цветом, так
как это свойство переопределено. Однако размер будет тот же.</B>
...
</BODY>
```

Вот что мы увидим:

**Этот текст будет отображен в соответствии с описанием в заголовке: красным цветом и размером шрифта в 120% от размера, принятого по умолчанию.**

**А этот текст будет отображаться синим цветом, так как это свойство переопределено. Однако размер будет тот же.**

CSS предоставляет нам еще одну замечательную возможность: *определение классов*. С помощью классов можно присваивать стили не всем одинаковым элементам страницы, а избирательно. Для того чтобы использовать класс, его необходимо вначале определить внутри элемента 'STYLE', а затем сослаться на этот класс в каком-либо элементе внутри 'BODY' с помощью атрибута CLASS:

```
<HEAD>
<STYLE TYPE="text/css">
.имя_класса {свойство: значение; свойство: значение}
</STYLE>
</HEAD>
<BODY>
...
<элемент CLASS="имя_класса">
что-то, что будет форматироваться в соответствии с
заданными в классе стилями
</элемент>
...
</BODY>
```

Пример:

```
<HEAD>
<STYLE TYPE="text/css">
.x {width: 160px}
.y {padding: 15px}
</STYLE>
</HEAD>
<BODY>
<TABLE BORDER=1 BGCOLOR=gray BORDERCOLOR=white>
<TR>
<TD>Ячейка1</TD>
<TD CLASS="x">Ячейка2</TD>
</TR>
<TR>
<TD CLASS="y">Ячейка3</TD>
</TR>
</TABLE>
</BODY>
```

## Результат:

Ячейка1	Ячейка2
Ячейка3	

В этом примере Ячейка 2 отформатирована в соответствии с классом “x”, для элементов которого установлена ширина в 160 пикселей, а Ячейка 3 — в соответствии с классом “y”, для всех элементов которого устанавливается внутренний отступ (“набивка”) в 15 пикселей. Ячейке 1 ни один из классов не присваивается, поэтому она форматируется в соответствии с общепринятыми стандартами HTML, а также согласно описанным в теге <TABLE> атрибутам. Часто классы используют вместе с тегом DIV. Этот тег никак не форматирует текст, а только выделяет в файле объект. Все, что находится в теге DIV, воспринимается браузером как один объект. Например, такую запись:

```
<DIV class=olive><B>это</B> очень <EM>важно  
</EM></DIV>
```

браузер должен вывести так:

**это очень *важно***

Кроме классов в CSS возможно использование идентификаторов (селекторов) ID.

ID — индивидуально именованный стиль. С его помощью можно создавать стилистические исключения среди элементов одного класса.

Идентификаторы используются в основном для придания одному или нескольким элементам одного класса индивидуальных свойств. Скажем, создан класс blue — синий курсив, но понадобился жирный подчеркнутый текст синим курсивом. Можно создать новый класс, однако проще описать ID. Например, “boldunderline”. И все элементы класса blue со значением ID “boldunderline” станут жирным подчеркнутым синим курсивом. Произойдет как

бы синтез свойств класса `blue` и идентификатора `boldunderline`.

Например:

```
<html>
<head>
<title> Пример CSS </title>
</head>
<style>
.blue {color:blue; font-style:italic}
#boldunderline {text-decoration:underline; font-weight:bold}
</style>
<body>
<p class="blue"> Здравствуйете, это моя домашняя страница. </p>
<p class="blue" id="boldunderline"> Пока еще в стадии разработки ... </p>
<p id="boldunderline">... Но скоро откроется </p>
</body>
</html>
```

#### 5.8.4. Встраивание в теги документа

Второй вариант, когда описание стиля располагается непосредственно внутри тега элемента, который вы описываете. Это делается с помощью параметра `STYLE`, используемого при применении CSS с большинством стандартных тегов HTML. Этот метод нежелателен, он приводит к потере одного из основных преимуществ CSS — возможности отделения информации от описания оформления информации. Впрочем, если необходимо описать лишь один элемент, этот вариант расположения описания стилей также вполне применим.

Его общий синтаксис таков:

*<элемент STYLE="свойство: значение; свойство: значение">текст или любой другой объект</элемент >*

Например, мы хотим, чтобы все абзацы на странице отображались шрифтом Times New Roman размером 20 пунктов темно-зеленого цвета. Для этого следует прописать атрибут `style` тега `<body>`, присвоив ему соответствующее значение. Синтаксис такой:

*<body style="font-family: 'Times New Roman', serif; font-size:20pt; color: darkgreen;">*

### 5.8.5. Импортирование

В теге `<STYLE>` можно импортировать внешнюю таблицу стилей с помощью свойства `@import` таблицы стилей:

```
import: url(mystyles.css).
```

Его следует задавать в начале стилевого блока или связываемой таблицы стилей перед заданием остальных правил. Значение свойства `@import` является URL файла таблицы стилей.

### 5.8.6. Связывание

Расположение описания стилей в отдельном файле имеет смысл в случае, если вы планируете применять эти стили к большему, чем одна, количеству страниц. Для этого нужно создать обычный текстовый файл, описать с помощью языка CSS необходимые стили, разместить этот файл на Web-сервере, а в коде Web-страниц, которые будут использовать стили из этого файла, нужно будет сделать ссылку на него. Делается это с помощью тега `LINK`, располагающегося внутри тега `HEAD` ваших страниц:

```
<LINK REL=STYLESHEET TYPE="text/css"
      HREF="URL">
```

Первые два параметра этого тега являются зарезервированными именами, требующимися для того, чтобы сообщить браузеру, что на этой страничке будет использоваться CSS. Третий параметр — `HREF= «URL»` — указывает на файл, который содержит описания стилей. Этот параметр должен содержать либо относительный путь к файлу — в случае, если он находится на том же сервере, что и документ, из которого к нему обращаются — или полный URL («`http://...`») в случае, если файл стилей находится на другом сервере.

Рассмотрим простой пример:

Содержимое файла `example.html`:

```
<HTML>
<HEAD>
<LINK REL=STYLESHEET TYPE="text/css" HREF="style.css">
<TITLE>Пример использования каскадных таблиц стилей</TITLE>
```

```
</HEAD>  
<BODY>  
<DIV class=first>Каскадные таблицы стилей </DIV>  
<DIV class=second>это</DIV>  
<DIV class=third>круто!</DIV>  
</BODY>  
</HTML>
```

Содержимое файла style.css:

```
body {font-family: Arial, sans-serif; text-align: center}  
.first {color: brown; margin-top: 60px; font-size: 40px;}  
.second {color: green; margin-top: -50px; font-size: 100px;}  
.third {color: black; margin-top: -80px; font-size: 120px; font-weight: bold}
```

А вот то, что вы должны увидеть на экране, запустив файл example.html, если у Вас включена поддержка каскадных таблиц стилей (рис. 5.31).

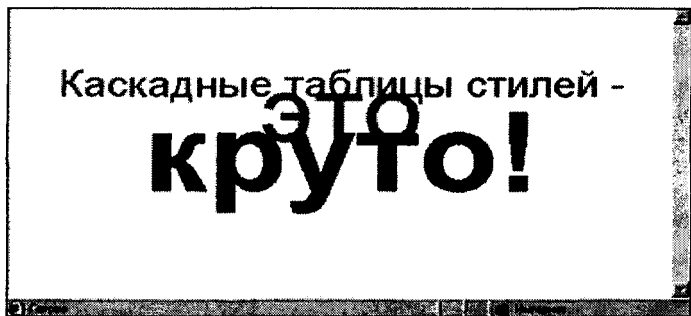


Рис. 5.31. Окно каскадной таблицы

### 5.7.8. Свойства элементов, управляемых с помощью CSS

В настоящее время язык CSS насчитывает довольно большое количество свойств элементов HTML, которыми он может управлять.

Здесь приводятся наиболее распространенные свойства для шрифтов, цветов и текста. Кроме того, существуют атрибуты и для таблиц, границ, курсоров, скроллов и т.д. (табл. 5.5—5.7).



## Свойства шрифтов

font-family	Используется для указания шрифта или шрифтового семейства, которым будет отображаться элемент <i>P {font-family: Times New Roman, sans-serif;}</i>
font-style	Задаёт способ начертания шрифта: normal — Нормальный (по умолчанию), italic - Курсив, oblique - Наклонный. <i>P {font-style: italic;}</i>
font-variant	Задаёт варианты начертания шрифта: normal — Нормальный (по умолчанию), small-caps — (Все буквы заглавные) <i>P {font-variant: small-caps;}</i>
font-weight	Определяет степень жирности шрифта с помощью параметров: normal, bold, bolder, lighter, 100, 200, 300, 400, 500, 600, 700, 800, 900 <i>B {font-weight: bolder;}</i>
font-size	Устанавливает размер шрифта. Параметр может указываться как в относительной (проценты), так и абсолютной величине (пункты, пиксели, сантиметры) <i>H1 {font-size: 200%;}</i> <i>H2 {font-size: 150px;}</i> <i>H3 {font-size: 400pt;}</i>

## Свойства цветов

color	Определяет цвет элемента <i>I {color: green;}</i>
background-color	Устанавливает цвет фона для элемента — именно для элемента, а не для странички <i>H4 {background-color: yellow;}</i>
background-image	Устанавливает или получает фоновый рисунок для элемента. Может принимать значения <i>none</i> — По умолчанию. Используется цвет родительского объекта <i>sUrl</i> — Задаётся абсолютный или относительный путь к рисунку <i>H3 {background-image: yellow;}</i>
background-attachment	Устанавливает или получает поведение фонового рисунка для элемента. Может принимать значения <i>scroll</i> — По умолчанию. Фоновый рисунок прокручивается вместе с элементом <i>fixed</i> — Фоновый рисунок не прокручивается вместе с элементом <i>body {background-attachment: fixed;}</i>

## Свойства текста

text-decoration	Устанавливает эффекты оформления шрифта, такие, как подчеркивание или зачеркнутый текст <i>H4 {text-decoration: underline;} — подчеркивание</i> <i>A {text-decoration: none;} - стандартный текст</i> <i>I {text-decoration: line-through;} — зачеркивание</i> <i>B {text-decoration: overline;} - надчеркивание</i>
text-transform	Задаёт преобразование регистра текста при отображении <i>H4 {text-transform: capitalize;} — Первая буква каждого слова преобразуется в заглавную</i> <i>A {text-transform: uppercase;} — Все буквы преобразуются в заглавные</i> <i>I {text-transform: lowercase;} — Все буквы преобразуются в строчные</i> <i>B {text-decoration: none;} — Отменяет установленные преобразования</i>
text-align	Определяет выравнивание элемента. Возможные значения: left, right, center, justify <i>P {text-align: justify}</i> <i>H5 {text-align: center}</i>
text-indent	Устанавливает отступ первой строки текста. Чаще всего используется для создания параграфов с табулированной первой строкой <i>P {text-indent: 50pt;}</i>
line-height	Управляет интервалами между строками текста <i>P {line-height: 50 %}</i>
word-spacing	Устанавливает интервалы между словами. Можно использовать отрицательные значения <i>P {word-spacing: 50 %}</i>
letter-spacing	Устанавливает интервалы между буквами <i>P {letter-spacing: 50 pt}</i>

---

## **Глава 6**

---

### **Создание Web-приложения в системе OFBIZ**

Мы рассмотрели основные программные технологии, используемые в системе OFBIZ для реализации графического Web-ориентированного интерфейса. К ним относится платформно независимый язык OFBIZ-Widgets и система генерации произвольных текстовых документов на основе шаблонов FreeMarker, используемая для динамичной подготовки HTML-страниц. В этой главе на основании полученных сведений мы создадим простое Web-приложение, в котором реализованы базовые функции информационной системы вуза. Несмотря на то, что мы практически не выйдем за рамки изученных ранее программных технологий и не будем использовать мощный инструмент ЦУС и ЦУЭД системы OFBIZ, в созданном графическом интерфейсе можно будет найти некоторые возможности, редко встречаемые даже на коммерческих сайтах. При создании Web-приложения будет сделана эталонная структура каталогов и конфигурационных файлов компонента системы OFBIZ, которые можно будет использовать в дальнейшей работе в качестве шаблона.

#### ***6.1. Постановка задачи***

Необходимо в виде самостоятельного компонента системы OFBIZ реализовать простой прототип Web-интерфейса информационной системы для ВУЗа. В задачи такой системы должны входить:

- ♦ информирование студентов и преподавателей о ближайших событиях, семинарах и других фактах из жизни ВУЗа (выдача новостей);

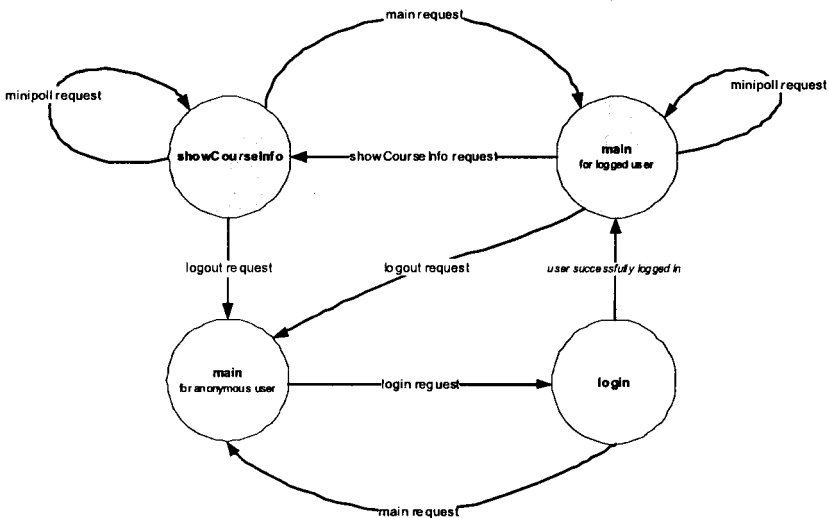
◆ проведение интерактивных опросов посетителей по различным темам (mini poll);

◆ доступ к материалам учебных курсов, где дается описание курса, его длительность и пр.;

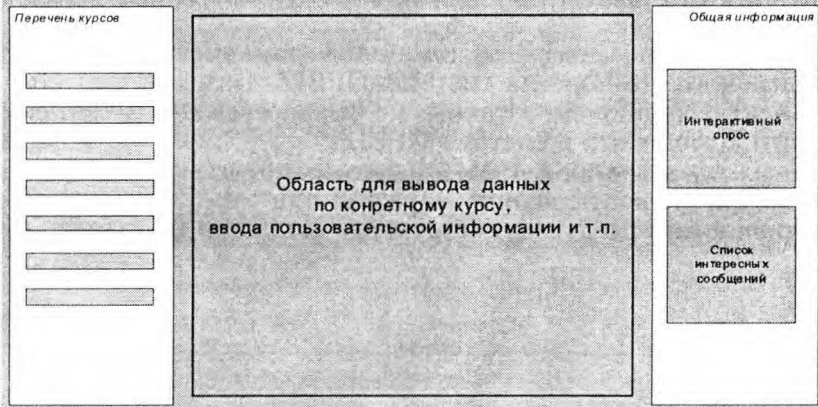
◆ регистрация пользователей в системе.

В соответствии с перечнем задач диаграмма переходов между состояниями (HTML-страницами) Web-приложения должна быть такой, как показано на рис. 6.1.

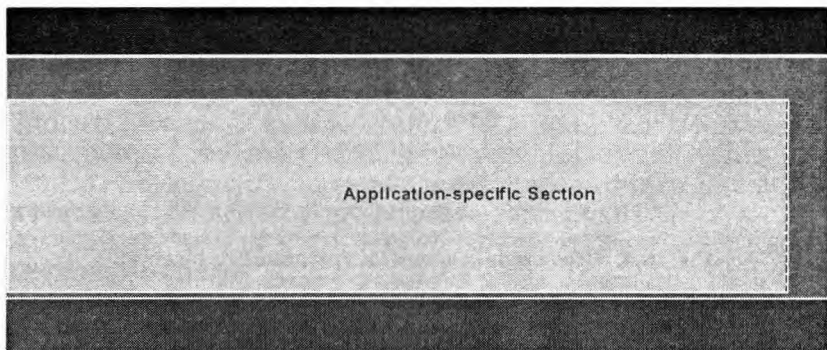
В Web-приложении создаются общедоступные страницы (выделены светло-серым цветом), не требующие предварительной регистрации, а также страницы, доступ к которым возможен лишь после введения правильного имени пользователя системы OFBIZ и пароля (например, admin-ofbiz). Поэтому нужно предусмотреть два варианта страницы с главным экраном (для анонимного и зарегистрированного пользователя) и особый экран для ввода имени и пароля. Кроме того, ожидается, что с приложением будут работать интернациональные пользователи из разных стран, поэтому потребуется обеспечить поддержку локализации.



**Рис. 6.1.** Диаграмма переходов между состояниями в приложении



**Рис. 6.2.** Эскиз Web-интерфейса



**Рис. 6.3.** Общая структура приложения с «экраном-декоратором»

Реализацию внешнего вида интерфейса планируется выполнить в соответствии со следующим эскизом (рис. 6.2).

В дальнейшем планируется разработка еще нескольких Web-приложений с близкой функциональностью, поэтому стоит сразу подумать о едином стиле оформления различных интерфейсов. Для этого предлагается использовать подход с «экранами-декораторами» и общую структуру приложения реализовать в виде, указанном на рис. 6.3.

### 6.1.1. Структура каталогов для компонента

Разработку Web-приложения будем вести в составе отдельного компонента системы OFBIZ. Назовем этот компонент «biminfo» и с таким же именем создадим каталог этого компонента в системе OFBIZ:

**%OFBIZ\_HOME%\specialized\biminfo.**

При проектировании и разработке компонента будем использовать следующую структуру каталогов (рис. 6.4):

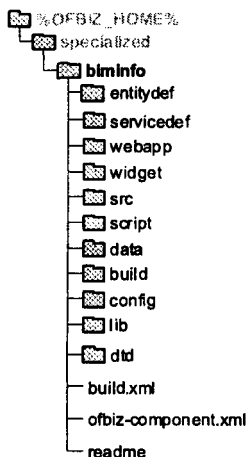


Рис. 6.4. Структура каталогов компонента biminfo

### 6.1.2. Адаптация стандартных конфигурационных файлов компонента

Перед тем, как непосредственно заняться разработкой графического Web-интерфейса, нужно провести адаптацию стандартных конфигурационных файлов с настройками регистрации компонента в системе OFBIZ<sup>26</sup>. К ним относятся файлы:

1. %OFBIZ\_HOME%\specialized\biminfo\ofbiz-component.xml;
2. %OFBIZ\_HOME%\specialized\biminfo\webapp\biminfo\WEB-INF\web.xml;

<sup>26</sup> Можно разместить каталог biminfo в %OFBIZ\_HOME%\hot-deploy, но это слишком просто.

```

<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="biminfo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/ofbiz-component.xsd">

  <!-- define resource loaders; most common is to use the component resource loader -->
  <resource-loader name="main" type="component"/>

  <!-- place the config directory on the classpath to access configuration files -->
  <classpath type="dir" location="config"/>
  <classpath type="dir" location="script"/>

  <webapp name="biminfo" title="BIM INFO"
server="default-server" location="webapp/biminfo" mount-point="/biminfo"/>
</ofbiz-component>

```

Рис. 6.5. Содержимое файла ofbiz-component.xml

### 3. %OFBIZ\_HOME%\specialized\component-load.xml.

В первом файле находятся общие настройки компонента, а во втором — специфичные параметры, относящиеся к конфигурации Web-интерфейса. Измененный вариант файла ofbiz-component.xml представлен на рис. 6.5.

Во втором файле (web.xml) изменения потребуются вносить лишь в начало файла. На рис. 6.6 представлен измененный вариант файла web.xml. Специфичные для нашего компонента значения выделены жирным шрифтом, а имена изменяемых параметров — подчеркнуты.

Видно, что, по сути, требуется изменить значения всего двух параметров:

```

<?xml version="1.0"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <display-name>Open For Business - BIM Info Component</display-name>
  <description>BIM Info Component of the Open For Business Project</description>
  <context-param>
    <param-name>webSiteId</param-name>
    <param-value>WebStore</param-value>
    <description>A unique ID used to look up the WebSite entity to get information about
catalogs, etc.</description>
  </context-param>

```

Рис. 6.6. Содержимое файла web.xml

```

<context-param>
  <param-name>entityDelegatorName</param-name>
  <param-value>default</param-value>
  <description>The Name of the Entity Delegator to use, defined in
entityengine.xml</description>
</context-param>

<context-param>
  <param-name>localDispatcherName</param-name>
  <param-value>biminfo</param-value>
  <description>A unique name used to identify/recognize the local dispatcher for the
ServiceEngi</description>
</context-param>

<filter>
  <filter-name>ContextFilter</filter-name>
  <display-name>ContextFilter</display-name>
  <filter-class>org.ofbiz.webapp.control.ContextFilter</filter-class>
  <init-param>
    <param-name>disableContextSecurity</param-name>
    <param-value>N</param-value>
  </init-param>
  <init-param>
    <param-name>allowedPaths</param-name>
    <param-value>
/control:/select:/index.html:/index.jsp:/default.html:/default.jsp:/images:/includes/maincss.css
  </param-value>
  </init-param>
  <init-param>
    <param-name>errorCode</param-name>
    <param-value>403</param-value>
  </init-param>
  <init-param>
    <param-name>redirectPath</param-name>
    <param-value>/control/main</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>ContextFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<listener>
  <listener-class>org.ofbiz.webapp.control.ControlEventListener</listener-class>
</listener>
<listener>
  <listener-class>org.ofbiz.securityext.login.LoginEventListener</listener-class>
</listener>
<servlet>
  <servlet-name>ControlServlet</servlet-name>
  <display-name>ControlServlet</display-name>

```

**Рис. 6.6.** Содержимое файла web.xml (продолжение)



```

<description>Main Control Servlet</description>
<servlet-class>org.ofbiz.webapp.control.ControlServlet</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>ControlServlet</servlet-name>
  <url-pattern>/control*</url-pattern>
</servlet-mapping>
<session-config>
  <session-timeout>60</session-timeout>
  <!-- in minutes -->
</session-config>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
  <welcome-file>index.html</welcome-file>
  <welcome-file>index.htm</welcome-file>
</welcome-file-list>
</web-app>

```

Рис. 6.6. Содержимое файла web.xml (окончание)

◆ `webSiteId` — идентификатор логического Web-сайта, к которому относится наше Web-приложение;

◆ `localDispatcherName` — уникальный для каждого компонента системы OFBIZ идентификатор, используемый при работе с ЦУЭД.

Чтобы наш компонент был автоматически загружен при старте системы OFBIZ, потребуется также внести дополнения (рис. 6.7) в файл `%OFBIZ_HOME%\specialized\component-load.xml`:

## 6.2. Определение правил обработки запросов в Web-приложении

Теперь можно, исходя из описания возможных состояний и переходов в приложении `biminfo` (рис. 6.1), определить все детали обрабатываемых в нашем приложении запросов. Как известно, вся логика обработки запросов в Web-приложении определяется в файле `controller.xml`<sup>27</sup>. На основании анализа структуры нашего приложения внесем в

<sup>27</sup> В нашем случае файл располагается в каталоге:

`%OFBIZ_HOME%\specialized\biminfo\webapp\biminfo\WEB-INF\`

```

<?xml version="1.0" encoding="UTF-8"?>
<component-loader
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/component-loader.xsd">
  <load-component component-location="{ofbiz.home}/specialized/community"/>
  <load-component component-location="{ofbiz.home}/specialized/opentravelsystem"/>

  <load-component component-location="{ofbiz.home}/specialized/biminfo"/>
</component-loader>

```

**Рис. 6.7.** Содержимое файла component-load.xml

Этот файл необходимые изменения, указав воспринимаемые запросы (request) и связанные с ними представления (view) (рис. 6.8):

```

<?xml version="1.0" encoding="UTF-8"?>
<site-conf xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/site-conf.xsd">
  <description>BIM Info Component Site Configuration File</description>
  <owner>The Open For Business Project Copyright (c) 2005</owner>
  <errorpage>/error/error.jsp</errorpage>
  <!-- event handlers -->
  <handler name="java" type="request" class="org.ofbiz.webapp.event.JavaEventHandler"/>
  <handler name="soap" type="request" class="org.ofbiz.webapp.event.SOAPEventHandler"/>
  <handler name="service" type="request"
class="org.ofbiz.webapp.event.ServiceEventHandler"/>
  <handler name="simple" type="request"
class="org.ofbiz.webapp.event.SimpleEventHandler"/>
  <!-- view handlers -->
  <handler name="jpublish" type="view"
class="org.ofbiz.webapp.view.JPublishViewHandler"/>
  <handler name="ftl" type="view" class="org.ofbiz.webapp.ftl.FreeMarkerViewHandler"/>
  <handler name="jsp" type="view" class="org.ofbiz.webapp.view.JspViewHandler"/>
  <handler name="http" type="view" class="org.ofbiz.webapp.view.HttpViewHandler"/>
  <handler name="datavision" type="view"
class="org.ofbiz.webapp.view.DataVisionViewHandler"/>
  <handler name="jasperpdf" type="view"
class="org.ofbiz.webapp.view.JasperReportsPdfViewHandler"/>
  <handler name="jasperxml" type="view"
class="org.ofbiz.webapp.view.JasperReportsXmlViewHandler"/>
  <handler name="screen" type="view"
class="org.ofbiz.widget.screen.ScreenWidgetViewHandler"/>
  <handler name="screenfop" type="view"
class="org.ofbiz.widget.screen.ScreenFopPdfViewHandler"/>
  <handler name="service-multi" type="request"
class="org.ofbiz.webapp.event.ServiceMultiEventHandler"/>
  <preprocessor>
  <!-- Events to run on every request before security (chains exempt) -->

```

**Рис. 6.8.** Содержимое файла controller.xml

```

<event type="java" path="org.ofbiz.securityext.login.LoginEvents"
invoke="checkExternalLoginKey"/>
</preprocessor>
<postprocessor>
  <!-- Events to run on every request after all other processing (chains exempt) -->
</postprocessor>

<!-- Localization Mappings -->
<request-map uri="setSessionLocale">
  <security https="false" auth="false"/>
  <event type="java" path="org.ofbiz.common.CommonEvents"
invoke="setSessionLocale"/>
  <response name="success" type="view" value="main"/>
  <response name="error" type="view" value="main"/>
</request-map>
<!-- Localization Mappings END -->

<!-- Security Mappings -->
<request-map uri="checkLogin" edit="false">
  <description>Verify a user is logged in.</description>
  <security https="false" auth="false"/>
  <event type="java" path="org.ofbiz.securityext.login.LoginEvents" invoke="checkLogin"/>
  <response name="success" type="view" value="main"/>
  <response name="error" type="view" value="login"/>
</request-map>

<request-map uri="login">
  <security https="false" auth="false"/>
  <event type="java" path="org.ofbiz.securityext.login.LoginEvents" invoke="login"/>
  <response name="success" type="view" value="main"/>
  <response name="error" type="view" value="login"/>
</request-map>

<request-map uri="logout">
  <security https="false" auth="true"/>
  <event type="java" path="org.ofbiz.securityext.login.LoginEvents" invoke="logout"/>
  <response name="success" type="request" value="main"/>
  <response name="error" type="view" value="main"/>
</request-map>

<request-map uri="autoLogout">
  <security https="false" auth="false"/>
  <event type="java" path="org.ofbiz.securityext.login.LoginEvents"
invoke="autoLoginRemove"/>
  <response name="success" type="request" value="checkLogin"/>
  <response name="error" type="view" value="main"/>
</request-map>
<!-- End of Security Mappings -->

<!-- Request Mappings -->

```

Рис. 6.8. Содержимое файла controller.xml (продолжение)

```

<request-map uri="main">
  <security https="false" auth="false"/>
  <response name="success" type="view" value="main"/>
</request-map>

<request-map uri="showCourseInfo">
  <security https="true" auth="true"/>
  <response name="success" type="view" value="showCourseInfo"/>
</request-map>

<request-map uri="minipoll">
  <security https="false" auth="false"/>
  <event type="service" invoke="createSurveyResponse"/>
  <response name="success" type="view" value="main"/>
  <response name="error" type="view" value="main"/>
</request-map>
<!-- end of request mappings -->
<!-- View Mappings -->
<view-map name="main" type="screen"
page="component://biminfo/widget/MainScreens.xml#main"/>
<view-map name="error" type="screen"
page="component://biminfo/widget/MainScreens.xml#error"/>
<view-map name="login" type="screen"
page="component://biminfo/widget/MainScreens.xml#login"/>
<view-map name="showCourseInfo" type="screen"
page="component://biminfo/widget/MainScreens.xml#showcourse"/>
<!-- end of view mappings -->
</site-conf>

```

Рис. 6.8. Содержимое файла controller.xml (окончание)

### 6.3. Разработка структуры Web-интерфейса

Наконец, после настройки конфигурационных файлов компонента и определения перечня распознаваемых запросов и представлений можно начать проектирование и реализацию Web-интерфейса нашего приложения с использованием языка OFBIZ-Widgets и шаблонов системы FreeMarker.

#### 6.3.1. Описание структуры экранов на языке OFBIZ-Widget

Определение экранов на языке OFBIZ-Widget, которые соответствуют представлениям, разместим в файле %OFBIZ\_HOME%\specialized\biminfo\widget>MainScreens.xml. В соответствии с эскизом интерфейса и структурой «экрана-декоратора» (рис. 6.2, 6.3) этот файл будет содержать описа-

ние главного «экрана-декоратора» с именем «main-decorator» и набор специализированных экранов-интерфейсов:

1) left — левая часть интерфейса, где отображается список курсов;

2) right — правая часть интерфейса, где отображаются опросы и новости (включает в себя экраны 6 и 7);

3) center — центральная часть интерфейса, где отображается информация по курсу;

4) main — главный экран при первом обращении, включающий экран 1;

5) showcource — главный экран с описанием курсов, включающий экраны 1, 2 и 3;

6) minipoll — внутренний экран, в который включается реализация опросов;

7) factoid — внутренний экран, в который включается реализация новостей;

8) login — главный экран для ввода имени пользователя и пароля.

### *6.3.2. Шаблоны системы FreeMarker*

Элементы интерфейса Web-приложения, требующие использования языка HTML, будут реализованы с помощью FTL-шаблонов системы FreeMarker.

Нам потребуется создать следующие шаблоны, расположенные в каталоге

`%OFBIZ_HOME%\specialized\biminfo\webapp\biminfo\includes:`

1) common/header.ftl — заголовок HTML-страницы с логотипом и списком выбора различных доступных локализаций;

2) common/appbar.ftl — панель инструментов приложения;

3) common/footer.ftl — «подвал» HTML-страницы;

4) appheader.ftl — заголовок с названием Web-приложения;

5) factoids.ftl — реализация новостей;

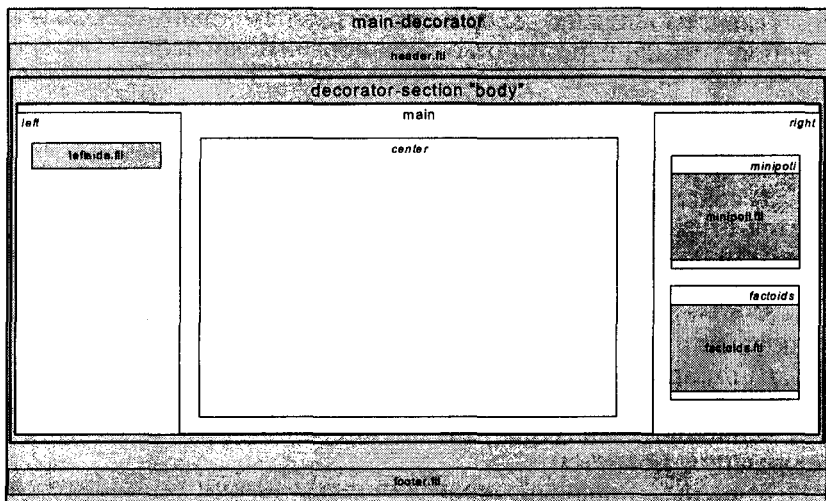
6) leftside.ftl — реализация заголовка левой части экрана;

7) minipoll.ftl — реализация опросов.

Перечисленные FTL-шаблоны включаются в состав соответствующих экранов с помощью тегов `<html-template>`.

### *6.3.3. Логическая структура файла MainScreens.xml*

Таким образом, объединенная логическая структура интерфейса нашего Web-приложения выглядит так, как показано на рис. 6.9.



**Рис. 6.9.** Логическая структура файла MainScreens.xml определяет все взаимосвязи между разными частями интерфейса

```
<screens ..... >
```

```
<screen name="left">
  <section>
    <widgets>
      <html-template
location="component://biminfo/webapp/biminfo/includes/leftside.ftl"/>
    </widgets>
  </section>
</screen>
```

```
<screen name="right">
  <section>
    <condition>
      <if-has-permission permission="CONTENTMGR" action="_VIEW"/>
    </condition>
    <widgets>
      <include-screen name="minipoll"/>
      <include-screen name="factoid"/>
    </widgets>
  </section>
</screen>
```

**Рис. 6.10.** Основа содержимого файла MainScreens.xml

<!-- DECORATOR SCREEN -->

```
<screen name="main-decorator">
  <section>
    <widgets>
      <html-template
location="component://biminfo/webapp/biminfo/includes/common/header.ftl"/>
      <html-template
location="component://biminfo/webapp/biminfo/includes/appheader.ftl"/>

      <include-screen name="left" location="{leftbarScreenLocation}"/>
      <include-screen name="right" location="{rightbarScreenLocation}"/>

      <decorator-section-include name="body"/>
    </widgets>
  </section>
</html-template>
```

```
location="component://biminfo/webapp/biminfo/includes/common/footer.ftl"/>
  </widgets>
</section>
</screen>
```

```
<screen name="center">
  <section>
    <widgets>
      <html-template location="component://biminfo/webapp/biminfo/main.ftl"/>
    </widgets>
  </section>
</screen>
```

```
<!-- MAIN SCREEN -->
<screen name="main">
  <section>
    <widgets>
      <decorator-screen name="main-decorator">
        <decorator-section name="body">
          <include-screen name="center"/>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
```

```
<screen name="showcourse">
  <section>
    <widgets>
      <decorator-screen name="main-decorator">
        <decorator-section name="body">
          </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
```

**Рис. 6.10.** Основа содержимого файла MainScreens.xml (продолжение)

```
<screen name="minipoll">
  <section>
    <widgets>
      <html-template
location="component://biminfo/webapp/biminfo/includes/minipoll.ftl"/>
    </widgets>
  </section>
</screen>
```

```
<screen name="factoid">
  <section>
    <widgets>
      <html-template
location="component://biminfo/webapp/biminfo/includes/factoids.ftl"/>
    </widgets>
  </section>
</screen>
```

```
<!-- login screens -->
<screen name="login">
  <section>
    <widgets>
      <decorator-screen name="main-decorator">
        <decorator-section name="body">
          <include-form name="login" location="component://biminfo/widget/forms.xml"/>
          <include-form name="forgotpassword"
location="component://biminfo/widget/forms.xml"/>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>
</screens>
```

**Рис. 6.10.** Основа содержимого файла MainScreens.xml (окончание)

Серым цветом на этом рисунке обозначен «экран-декоратор» и FTL-шаблоны, а белым — различные специфичные экраны 1—7. Приведем здесь же «выжимку» из файла MainScreens.xml, чтобы было видно соответствие между логической структурой из рис. 6.9 и структурой XML-тегов. Заметим, что в этой «выжимке» присутствует только часть реального содержимого файла MainScreens.xml (рис. 6.10).



A screenshot of a login form. It features a 'Username' label followed by a text input field containing the text 'admin'. Below it is a 'Password' label followed by an empty text input field. At the bottom of the form is a button labeled 'Login'.

A screenshot of a password recovery form. It has a 'User Name' label followed by an empty text input field. Below the input field are two buttons: 'Get Password Hint' on the left and 'Email to send password' on the right.

Рис. 6.11. Используемые в составе приложения формы

#### 6.4. Формы на языке OFBIZ-Widget

В описании экрана регистрации login мы воспользуемся механизмом форм языка OFBIZ-Widget. Определение форм разместим в файле

```
%OFBIZ_HOME%\specialized\biminfo\widget\forms.xml.
```

В нем потребуется создать описание двух форм: login и forgotpassword (рис. 6.11).

Полный файл с описанием форм показан на рис. 6.12.

Структурный каркас нашего Web-приложения biminfo создан. В определении экранов можно добавлять необходимое содержание и функциональность, а также создавать FTL-шаблоны. Но перед тем, как продолжить разработку интерфейса в этом направлении, нам потребуется знакомство с назначением и основными приемами работы в системе управления содержанием Web-сайта (*Content Management System*), входящей в состав OFBIZ. Здесь мы впервые воспользуемся функциональностью «верхнего» уровня архитектуры OFBIZ (уровень «Applications») и увидим, что активное применение стандартных возможностей этой системы позволит намного сократить усилия и время на разработку частей интерфейса, а также сделать содержимое нашего приложения более динамичным.

```

<?xml version="1.0" encoding="UTF-8"?>
<forms xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/widget-form.xsd">
  <form name="login" target="login" title="" type="single"
    default-title-style="tableheadtext" default-tooltip-style="tabletext" default-widget-
style="inputBox">
    <field name="USERNAME" title="Username" entry-name="userLoginId" map-
name="autoUserLogin">
      <text size="20"/>
    </field>
    <field name="notUser" title="" use-when="0 &lt;
&quot;${autoUserLogin.userLoginId}&quot;.length() ">
      <hyperlink also-hidden="false" description="not a &nbsp;${autoUserLogin.userLoginId}"
target="${autoLogoutUrl}" target-type="intra-app"/>
    </field>
    <field name="PASSWORD" title="Password">
      <password size="20"/>
    </field>
    <field name="submitButton" title="Login" widget-style="standardSubmit">
      <submit button-type="button"/>
    </field>
    <field name="previousParams">
      <hidden/>
    </field>
  </form>
  <form name="forgotpassword" target="/ecommerce/control/forgotpassword${previousParams}"
target-type="inter-app" title="" type="single" default-title-style="tableheadtext" default-tooltip-
style="tabletext" default-widget-style="inputBox">
    <field name="USERNAME" title="User Name" entry-name="userName">
      <text size="20"/>
    </field>
    <field name="GET_PASSWORD_HINT" title="Get Password Hint" widget-
style="smallSubmit" position="1">
      <submit button-type="button"/>
    </field>
    <field name="GET_EMAIL_PASSWORD" title="Email to send password" widget-
style="smallSubmit" position="2">
      <submit button-type="button"/>
    </field>
  </form>
</forms>

```

Рис. 6.12. Содержимое файла forms.xml

## **6.5. Назначение и приемы использования системы управления содержанием (Content Manager) в OFBIZ**

В начальный период зарождения Web-приложений большая часть содержимого даже для коммерческих Web-сайтов создавалась один раз на этапе его проектирования. Неудивительно, что с течением времени информация на них переставала быть актуальной. С такой ситуацией можно было мириться, пока количество посетителей сайта было невелико и они не представляли большого интереса с маркетинговой точки зрения.

Однако к настоящему времени ситуация совершенно изменилась. Реклама и продажа товаров через Интернет стали одной из важнейших функций практически любого предприятия. К информации, размещенной на Web-сайте компании, обращаются ежедневно сотни, а то и тысячи пользователей из разных стран, с различными информационными потребностями, социальными стереотипами и покупательской способностью. Поэтому информационное содержание Web-сайта (часто для этого используется термин Content) становится действительно серьезным маркетинговым инструментом, от эффективного использования которого во многом зависит успех продаж не только в Интернете, но и через обычные розничные или оптовые каналы.

Обязательное требование поддержки актуальности содержания Web-сайта и, как следствие, — сильная взаимосвязь с различными данными предприятия влечет за собой необходимость включения функций разработки и управления содержанием Web-сайта в состав КИС. В современном маркетинге уже сложились определенные правила структурирования и подачи информационных материалов о продуктах и деятельности компании. Наиболее значимыми из них являются:

- ◆ целевая направленность информации на определенную группу покупателей;
- ◆ активное использование различных скидок при покупке определенного набора товаров, покупки в определенный срок и т.п. (так называемый promotion);
- ◆ сохранение и тщательный анализ профиля покупателя (история покупок, стабильные маршруты просмотра

информации и т.п.) для определения его личных предпочтений и выделения групп (категорий) покупателей с близкими параметрами;

♦ предоставление по запросу пользователей разноплановых обзоров и аналитических материалов по любому продаваемому продукту.

Несомненно, что КИС должна поддерживать эффективное выполнение этих правил при разработке и управлении содержанием Web-сайта и обеспечивать:

1) быстрое обновление внешнего вида или информации на сайте без ручной переделки множества HTML-страниц и привлечения дизайнеров;

2) простой механизм динамического включения в HTML-страницу актуальной информации о ценах на продукцию, сведениях о наличии на складе и т.п.;

3) выдачу одной и той же информации на различных языках и в разных форматах;

4) сокращение избыточности и недопущение противоречивости информации на Web-сайте за счет возможности использования в разных страницах ссылок на один и тот же экземпляр документа или сообщения;

5) ведение информационного каталога и поиск различных документов и сообщений, размещенных на сайте;

6) сбор, хранение и поиск персонального профиля для зарегистрированных посетителей сайта;

7) выдачу посетителю сайта информационного содержания (включая различную отпускную цену), зависящего от его профиля и истории прежних посещений сайта компании (например, каталоги продукции и обзоры по определенной, интересующей данного пользователя тематике);

8) эффективную обратную связь с потенциальными потребителями за счет проведения на Web-сайте различных опросов, ведения интерактивных форумов и т.п., с последующей статической обработкой.

В системе OFBIZ реализован компонент Content Management System (CMS) — «система управления содержанием», который решает многие перечисленные задачи. Рассмотрим основные понятия, используемые в этом компоненте, и принципы его работы.

Среди целей использования системы управления содер-

жанием одной из важнейших является повышение коэффициента повторного использования информации, размещаемой на Web-сайте. Это позволяет однажды размещенную под управление CMS информацию (документ, новость, факт и т.п.) использовать в составе различных страниц. Причем, изменив содержимое один раз, можно быть полностью уверенным, что во всех страницах, где оно используется, автоматически произойдут нужные изменения.

Чтобы обеспечить такую функциональность в CMS OFBIZ для хранения информации и ее структурирования совместно используются два понятия — *содержание* (Content) и *информационный ресурс* (DataResource), а для управления информацией предоставляется специальное графическое Web-приложение Content Manager.

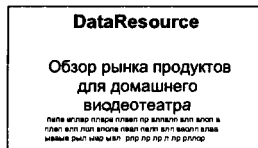
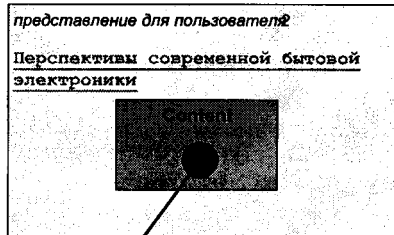
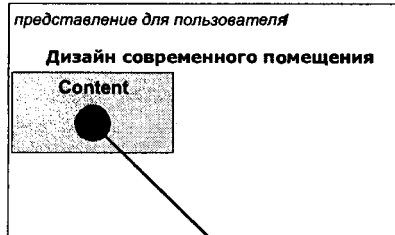
Содержание можно рассматривать как пустой контейнер для информации, который можно вставить в состав любого описания экрана на языке OFBIZ-Widget или в FTL-шаблон системы FreeMarker. Каждый экземпляр содержания сам по себе не содержит полезной информации для посетителей сайта, а ссылается на определенный экземпляр информационного ресурса, причем различные экземпляры содержания могут ссылаться на один и тот же информационный ресурс (рис. 6.13).

Такой подход отдаленно напоминает опосредованный вызов методов класса через методы интерфейса в языке Java: не меняя описание экземпляра содержания в составе экрана или HTML-страницы, мы можем просто изменить соответствующий информационный ресурс в графическом приложении администратора, и информационное наполнение содержания поменяется. Причем оно изменится во всех экземплярах содержания, которые ссылаются на измененный экземпляр информационного ресурса.

На рис. 6.14 представлена UML-диаграмма, в которой определены основные атрибуты и связи между наиболее важными классами системы управления содержанием OFBIZ<sup>28</sup>. На этой диаграмме вы встретитесь с новым выразительным средством языка UML: прерывистой лини-

---

<sup>28</sup> В приложении, те кто знаком с основами базы данных, могут изучить более подробную диаграмму, определяющую взаимосвязи между соответствующими элементами данных.



**Рис. 6.13.** Взаимосвязь между содержанием (Content) и информационным ресурсом (DataResource) в системе OFBIZ

ей, соединяющей определенный класс и сплошную линию связи между двумя другими классами. Таким образом, в языке UML обозначаются атрибуты связи между классами. Обращаем внимание на то, что атрибуты классов указаны только для важнейших классов. Каждый экземпляр класса содержание (Content) имеет уникальный идентификатор (contentId) и принадлежит к определенному логическому типу (экземпляр класса ContentType), связь с которым осуществляется через атрибут content-TypeId.

К наиболее часто используемым логическим типам относятся:

- ◆ Document — ссылка на определенный информационный ресурс или объединение нескольких подчиненных экземпляров содержания (связь осуществляется через ContentAssoc);

- ◆ Publish Point — место публикации связанного содержимого;

- ◆ Annotation — аннотация;

- ◆ Content List — список содержания;

- ◆ Tree Root — корень дерева, определяющего взаимосвязь экземпляров содержания;

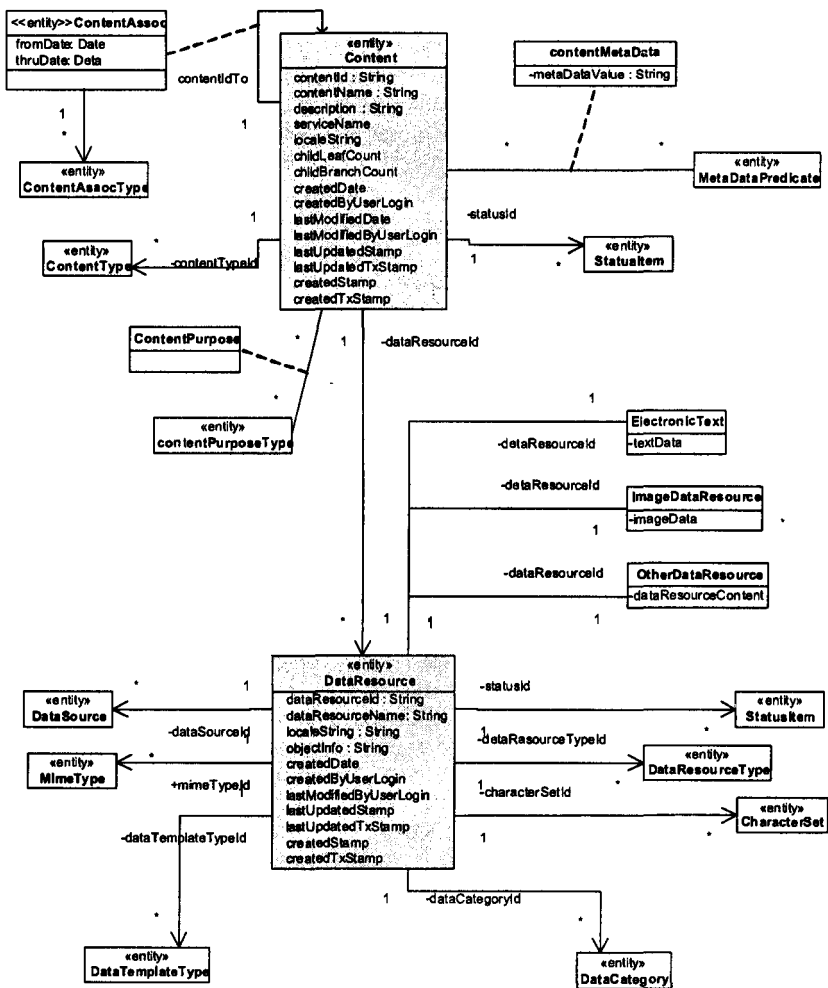


Рис. 6.14. Модель данных системы управления содержанием

- ◆ Graph Root — корень произвольного графа, определяющего взаимосвязь экземпляров содержания;
- ◆ Structure Node — нелистовой узел графа, определяющего взаимосвязь экземпляров содержания;

◆ **Structure Leaf** — листовой узел графа, определяющего взаимосвязь экземпляров содержания.

Через экземпляры класса **ContentPurpose** осуществляется связь между экземпляром содержания и определенным экземпляром класса **contentPurpose-Type**. Определение связи служит для пояснения назначения этого экземпляра содержания в составе системы. Обычно возможными типами назначения являются:

- ◆ **Article** — статья;
- ◆ **Message** — сообщение;
- ◆ **Feedback** — ответ;
- ◆ **Comment** — комментарий.

Среди других атрибутов и связей содержания с другими данными укажем на:

◆ **Locale String** — строка, определяющая локализацию данного экземпляра содержимого (например, **ru\_RU**);

◆ **Owner Content ID** — идентификатор владельца данного экземпляра содержания;

◆ **Data Source Id** — из какого источника данных (БД) получать информацию.

Таким образом, в системе **OFBIZ** можно определить все значимые характеристики содержания, а потом проводить по их значению быстрый поиск и классификацию.

### *6.5.1. Связь элементов содержания (Association)*

В большинстве практических случаев между различными экземплярами содержания существует логическая связь. Примерами такой связи являются:

◆ различные части текста в составе одного документа (введение, главная часть, заключение);

◆ различные сообщения в составе одного форума;

◆ различные письма в одном почтовом ящике;

◆ и т.п.

В системе **OFBIZ** можно явно определить связь между различными экземплярами содержания. В ходе динамической генерации интерфейса из описания экрана или **FTL**-шаблона, исходно имея доступ лишь к родительскому содержанию, можно обратиться к любому логически подчиненному экземпляру содержания, а затем к конкретному информа-



ционному наполнению — экземплярам соответствующих информационных ресурсов.

Из рис. 6.14 видно, что связь между различными экземплярами содержания реализуется с помощью атрибута `ContentIdTo` в классе `Content` и вспомогательного класса `ContentAssoc`, хранящего атрибуты связи (например, временной интервал, в течение которого действует связь, тип связи). Каждый тип связи имеет вполне определенное значение и на самом деле является ссылкой на какой-то предопределенный экземпляр класса `ContentAssocType`.

Например, если какой-то экземпляр содержания связан с другими, подчиненными экземплярами содержания, и этой связи присвоен тип «`PUBLISH_LINK (SUBSITE)`», то смысл этого содержания-владельца будет заключаться в том, что оно обозначает «место для публикации» различных информационных сообщений на сайте. Каждое отдельное сообщение является подчиненным экземпляром содержания и имеет тип «`Document`». Конкретным случаем реализации такой связи является область интерфейса «`Browse Content`» в стандартном Web-приложении `escommerce`. Экземпляр содержания, который эту область реализует и включает в себя все информационные сообщения, имеет значение атрибута `contentId` «`WebStoreCONTENT`». По этому значению можно провести поиск и изучить детальное определение этого экземпляра содержания и все его связи с подчиненными экземплярами содержания в стандартном Web-приложении `content`.

### 6.5.2. *MetaData*

Мета-данные позволяют формальным образом определить значение (семантику) экземпляра содержания. Это полезно для проведения поиска или сложного выбора содержания по профилю пользователя. Задание мета-данных в системе `OFBIZ` проводится путем определения значений для предопределенных атрибутов.

### 6.5.3. *DataResource*

Реальная информация для конечного пользователя располагается в информационном ресурсе (`DataResource`). Реализация в системе `OFBIZ` выполнена таким образом, что

один и тот же экземпляр информационного ресурса может повторно использоваться в составе различных экземпляров содержания (Content). Сами по себе экземпляры информационного ресурса включаются в состав экрана или FTL-шаблона редко, кроме тех, которые хранят графические изображения.

Наиболее значимые характеристики информационного ресурса определены в двух атрибутах класса `DataResource`:

- ◆ `mimeTypeId` — формат информации (например, `text/html` или `image/jpeg`);

- ◆ `dataResourceTypeId` — определение способа хранения информации в системе OFBIZ. Могут использоваться типы информационных ресурсов, перечисленные ниже:

- `LongText` — текст произвольного размера (например, фрагмент HTML-страницы), хранящийся в ЦУЭД (в базе данных);

- `Short Text` — текст до 255 символов, находящийся прямо в атрибуте `Object Info`;

- `Image` — графическое изображение, хранящееся в ЦУЭД (базе данных);

- `Local File` — произвольный текстовый файл, размещенный относительно корневого каталога системы OFBIZ (`%OFBIZ_HOME%`), текущего каталога web-приложения или по абсолютному пути;

- `HyperLink` — URL внешнего ресурса в Интернете, значение задается прямо в атрибуте `Object Info`. При генерации подставляется только адрес ресурса (текст URI);

- `URL Resource` — URL внешнего ресурса в Интернете, значение задается прямо в атрибуте `Object Info`. При генерации подставляется реальное содержимое ресурса;

- `Local Binary File` — произвольный файл, размещенный относительно корневого каталога системы OFBIZ (`%OFBIZ_HOME%`), текущего каталога web-приложения или по абсолютному пути.

Заметим, что подготовка информационных ресурсов может происходить динамически, на основании заранее созданных шаблонов на языках OFBIZ-Widget или FreeMarker. Для этого нужно определить значение атрибута `Data template Type Id`:

◆ FreeMarker — используется шаблон системы FreeMarker;

◆ XLST — используется преобразование XML с помощью стандарта XSLT;

◆ Screen Widget — используется OFBIZ-Widgets.

Среди других атрибутов информационного ресурса укажем на:

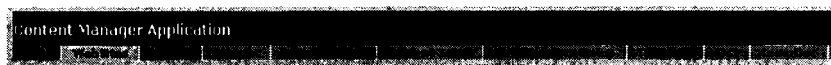
◆ LocaleString — строка, определяющая локализацию данного экземпляра информационного ресурса (например, ru\_RU);

◆ Charset — строка, определяющая кодировку данного экземпляра информационного ресурса (например, Windows-1251).

В стандартном Web-приложении ContentManager возможно создание и редактирование экземпляров информационного ресурса, а также поиск экземпляров содержания (Content), использующих данный информационный ресурс

## **6.6. Графические приложения для работы с содержимым**

В системе OFBIZ управление содержанием сайта, создание и редактирование экземпляров содержания или информационных ресурсов происходит с помощью стандартного Web-приложения ContentManager (рис. 6.15) (если OFBIZ запущен на локальной машине, то адрес в браузере — <http://127.0.0.1:8080/content/>).



**Рис. 6.15.** Панель инструментов в приложении Content Manager

Это приложение имеет следующие возможности (в порядке следования команд меню).

1. Настройка статистики обращений к различным ресурсам на сайте и создание логических Web-сайтов (*WebSites*).

2. Редактирование и создание интерактивных опросов (*Surveys*).

3. Поиск, редактирование и создание новых экземпляров содержания (*Content*).

4. Поиск, редактирование и создание новых информационных элементов (*Data Resource*).

5. Редактирование и создание новых типов содержания, типов связи, доступных действий с содержанием и т.п. (*Content Setup*).

6. Редактирование и создание новых типов информационных ресурсов, их категорий, доступных параметров метаданных, дополнительных атрибутов и т.п. (*Data Resource Setup*).

7. Поиск, редактирование и создание новых шаблонов (*Template*).

8. Поиск, редактирование и создание новых взаимосвязанных пар «содержание—информационный ресурс», управление связью между разными экземплярами содержания, модерирование форумов, индексирование содержания, поиск по сайту (*CMS*).

9. Создание и редактирование документов со сложной структурой (*CompDoc*).

Большинство экранов приложения Content интуитивно понятно, поэтому мы ограничимся обзором лишь наиболее важных интерфейсов.

◆ В интерфейсе Content/Association определяются правила связи различных экземпляров содержания между собой.

◆ В интерфейсе Content/Role с экземпляром содержания может быть связан отдельный пользователь системы OFBIZ, для которого определена какая-либо роль (создатель, редактор и т.п.).

◆ Интерфейс DataResource/Browse выдает пользователю иерархически-организованный тематический каталог информационных ресурсов (к сожалению, функциональность полностью не реализована).

◆ В интерфейсе ContentSetup можно изменять существующие типы содержания, виды связи различных экземпляров содержания между собой, типы прав доступа и другую служебную информацию, которая используется в системе CMS при работе с содержанием.

◆ Подобным образом в интерфейсе DataResourceSetup можно изменять различные служебные перечисления (типы информационных ресурсов, известные в системе кодировки

и т.п.), которые используются в системе CMS при работе с информационными ресурсами.

◆ Интерфейс Template примечается для работы с шаблонами — часто используемыми фрагментами описаний HTML-страницы на языке системы FreeMarker или иной системы динамичной генерации содержимого.

Комплексные задачи по организации связи экземпляров содержания с информационными ресурсами выполняются с помощью интерфейса CMS (рис. 6.16). Здесь же можно управлять структурой и содержанием интерактивных форумов, ведущихся на web-сайте (рис. 6.16, 6.17).

При выборе определенного экземпляра содержания из таблицы на рис. 6.16 пользователь получает возможность редактирования всех параметров связанных между собой экземпляров содержания и информационных ресурсов, а также создания новых связей (рис. 6.18).

В интерфейсе CompDoc реализованы функции создания, редактирования и управления версиями структурированных документов. По идее разработчиков пользователь сможет выполнять такие действия:

◆ создавать и редактировать иерархически организованные шаблоны структурированных документов. Каждый такой шаблон является деревом, в котором вершины обозначают различные типы информационных ресурсов (текст, документ Word, документ PDF и т.п.);

◆ создавать и редактировать экземпляры структурированных документов на основе одного из существующих шаблонов. Пользователь имеет возможность просматривать структуру экземпляра, добавлять или удалять составные части;

◆ вести историю создания шаблонов и экземпляров составных документов;

◆ подтверждать или отменять проведенные другими пользователями изменения в структуре шаблонов или в содержании экземпляров документов;

◆ генерировать на основе экземпляра структурированного документа HTML-страницу или PDF-файл.

К сожалению, реализация этих возможностей начата только недавно и еще далека от завершения.

Main Web Sites Home Content Data Resources Content Manager CMS Comp Doc

Content Sub Sites Index Search

Publish-to content   Equals  Begins With  Contains  Is Empty  Ignore Case

Map Key   Equals  Begins With  Contains  Is Empty  Ignore Case

Assoc Type Id   Equals  Begins With  Contains  Is Empty  Ignore Case

From Date   Equals  Same Day  Greater Than From Day Start  Greater Than   Less Than  Up To Day  Up Thru Day  Is Empty

Content Id   Equals  Begins With  Contains  Is Empty  Ignore Case

Data Resource Id   Equals  Begins With  Contains  Is Empty  Ignore Case

Name   Equals  Begins With  Contains  Is Empty  Ignore Case

Edit Content	Publish-to content	Map Key	From Date	Content Id	Data Resource Id	Name
<input type="button" value="Edit"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	BIM_COURSES	<input type="text"/>	<input type="text"/>

Рис. 6.16. Инструмент поиска взаимосвязанных экземпляров содержания и информационных ресурсов в приложении Content Manager

Main Web Sites Home Content Data Resources Content Manager CMS Comp Doc

Content Sub Sites Index Search

WebStoreFORUM

- Ask the Experts
- Gizmos
- Widgets

W3C CSS  W3C XHTML 1.0

Copyright (c) 2001-2006 The Open For Business Project - www.ofbiz.org  
Powered By OFBiz

Рис. 6.17. Инструмент управления форумами в приложении Content Manager

Content | Sub Sites | Index | Search

Go to Find

### ContentAssoc

**Область определения параметров связи между экземплярами Содержания** 1

Content Id To: BIM\_COURSES  
 MapKey: DESCRIPTION  
 ContentAssocTypeId: DESCRIPTION  
 ContentAssocPredcataId: \_\_\_\_\_  
 FromDate: 2005-03-02 17:58:34.312 TimeDate: \_\_\_\_\_

---

### Content

**Область определения параметров Содержания** 2

Content Id: BIM\_COURSE\_3  
 Template Data Resource Id: \_\_\_\_\_  
 Content Type Id: Document  
 Title: \_\_\_\_\_  
 Mime Type Id: \_\_\_\_\_  
 Status Id: \_\_\_\_\_

Owner Content Id: BIM\_COURSES  
 Description: Software Engineering  
 Character Set Id: \_\_\_\_\_  
 Privilege Enum Id: \_\_\_\_\_

---

### DataResource

**Область определения параметров Инф. ресурса** 3

Data Resource Type Id: Long Text  
 File Path: \_\_\_\_\_  
 Mime Type Id: text/html - HTML Text  
 Character Set Id: \_\_\_\_\_  
 Data Source: \_\_\_\_\_  
 Category: \_\_\_\_\_

#### Text

Text: 

```
<h3>TEST OF HTML TEXT</h3><p>this is a course of HTML-formatting</p><p><em><font color="#33cc33">Number of points - 20</font></em></p>
```

#### Image

Image Data: \_\_\_\_\_

Data Created By User: admin  
 Data Modified By User: admin

Data Create Date: 2005-03-02 17:58:33.801  
 Data Modified Date: 2005-03-14 19:18:07.179

TEST OF HTML TEXT

this is a course of HTML-formatting

Number of points - 20

**Рис. 6.18.** Основной экран для работы в приложении Content Manager (цифры определяют порядок работы в различных группах полей на этом экране)

## 6.7. *Использование возможностей CMS для реализации информационной системы*

В целях структурированного хранения информации о курсах в составе разрабатываемой информационной системы мы будем использовать следующие правила организации содержания в системе CMS:

1) каждый учебный курс представляется отдельным экземпляром содержания с уникальным названием и типом «Document»;

2) все экземпляры содержания, обозначающие курсы, связаны с единственным «родительским» экземпляром. Этот «родительский» экземпляр содержания имеет предопределенное имя `BIM_COURSES` и тип «Publish Point» (`WEB_SITE_PUB_PT`);

3) связь курсов и «родительского» содержания `BIM_COURSES` имеет тип связи `DESCRIPTION`;

4) каждый экземпляр содержания, описывающий учебный курс, должен содержать заполненные поля краткого описания и типа локализации (страна и язык);

5) каждый экземпляр содержания, описывающий учебный курс, должен быть связан с одним экземпляром информационного ресурса, который хранит подробное описание курса.

На рис. 6.18 показан пример правильно сформированного описания одного из предлагаемых курсов. Это курс «Software Engineering», который читается на английском языке (на рисунке не видно поля `Locale`, в котором хранится значение `en_US`). Подробное описание курса выполнено в форме фрагмента на HTML-странице.

### 6.7.1. *Элементы языка OFBIZ-Widget и FTL-теги для доступа к содержимому Content*

Вероятно, на этот момент главный вопрос по Системе управления содержанием звучит так — как можно использовать определенные ранее экземпляры содержания при разработке описания интерфейса на языке OFBIZ-Widgets или в FTL-шаблоне системы FreeMarker?

Для включения определенного экземпляра содержания и связанного с ним информационного ресурса в состав



описания экрана на языке OFBIZ-Widget можно использовать теги <content> и <sub-content>. Полный перечень атрибутов этих элементов приведен в главе 5. Например:

```
<content content-id="{thisContentId}" xml-escape="false" />
<sub-content assoc-name="SUMMARY" content-id="{blog.contentId}"></sub-content>
```

Если требуется включить экземпляр содержания в FTL-шаблон, то необходимо воспользоваться встроенными тегами renderSubContent, limitedSub-Content, loopSubContent Cache. Например:

```
<@renderSubContent subContentId=contentId/>
<@limitedSubContent subContentId=factoidRootId
  viewIndex=0
  viewSize=9999
  orderBy="contentName"
  contentAssocTypeId="SUBSITE"
  limitSize="2">
  <@renderSubContentCache subContentId=subContentId/>
</@limitedSubContent >
<@loopSubContentCache subContentId=forumRootId
  viewIndex=0
  viewSize=9999
  orderBy="contentName"
  contentAssocTypeId="SUBSITE" >
  <a href="{@ofbizUrl}/showcontenttree?contentId={subContentId}</@ofbizUrl"
  >
    ${content.contentName}
  </a>
  <#assign count_1=count_1 + 1 />
</@loopSubContentCache >
```

### 6.7.2. Использование Survey

Специальной разновидностью информационного ресурса в системе OFBIZ является интерактивный опрос (Survey). В интерактивном опросе определяется взаимосвязанная логическая структура в виде перечня predetermined-ных вопросов различного типа (рис. 6.19).

Администратор системы OFBIZ имеет возможность динамично компоновать новые интерактивные опросы с различными перечнями вопросов (или изменять существующие), объединять запросы в группы и связывать опросы с определенным логическим Web-сайтом в системе управления содержанием. Например, на рис. 6.20 показано описание большого опроса «Tell Us about your on-line shopping patterns» с предыдущего рисунка.

Tell us about your on-line shopping patterns

Your answers will help us serve you better



How often do you shop on-line?  \*

Have you shopped here before?  \*

How would you rate this store for products?  [optional]

How would you rate this store for on-line functionality?  \*

Do you have a picture you would like to attach?   [optional]

What is your favorite on-line store?  \*

Comments:  [optional]

Рис. 6.19. Внешний вид интерактивных опросов в приложении ecommerce

В системе OFBIZ интерактивные опросы активно используются в приложении «Интернет-магазин» (eCommerce). Поэтому опросы обычно связаны с определенным складом (в терминах OFBIZ — Store) и с логическим Web-сайтом<sup>29</sup>, который относится к этому складу. В web-приложении доступны опросы только для того Web-сайта, чей идентификатор указан в конфигурационном файле web-приложения web.xml (параметр WebSiteId). При установке системы OFBIZ создается несколько интерактивных опросов. Все они относятся к логическому Web-сайту WebStore.

Связь опроса с определенным складом происходит в Web-приложении Catalog (рис. 6.21).

<sup>29</sup> Понятие логического Web-сайта служит для тематической классификации множества различных экземпляров содержания и информационных ресурсов. В составе системы OFBIZ, физически реализующей один Web-сайт, логически может быть определено несколько логических Web-сайтов. К каждому из них принадлежат ресурсы близкого содержания (например, содержание одного магазина или одного подразделения компании и т.п.).

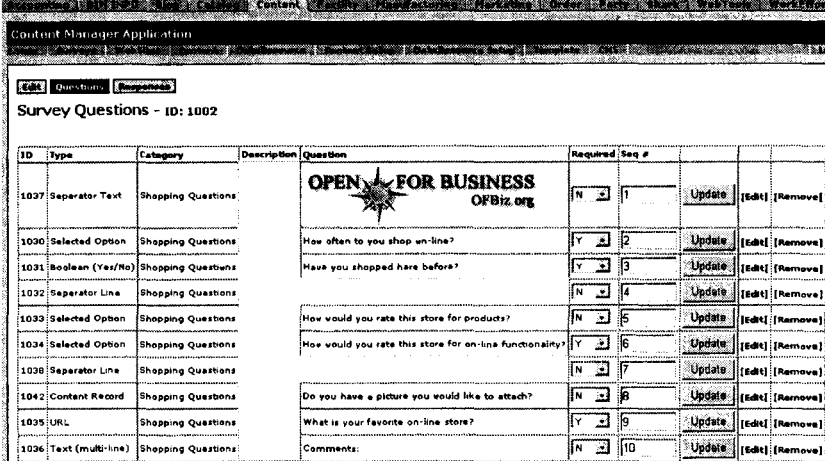


Рис. 6.20. Интерфейс для разработки описания интерактивного опроса

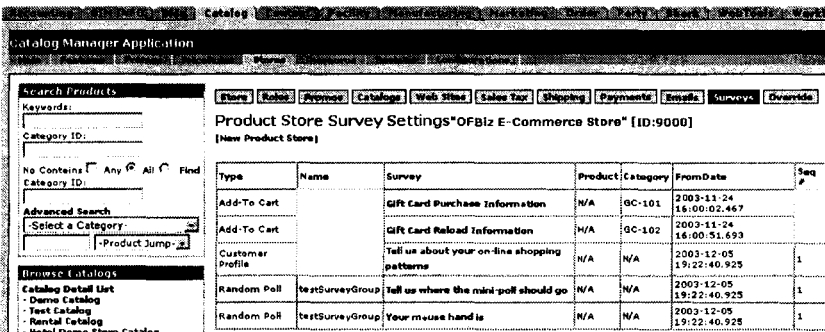


Рис. 6.21. Связь интерактивных опросов с определенным складом

Связывая опрос со складом, вы должны указать тип опроса, имя группы, к которой будет принадлежать опрос, а также шаблоны, на основе которых будет генерироваться визуальное представление опроса и результата (рис. 6.22).

В качестве значения полей «Survey Template Path» и «Result Template Path» можно использовать уже существующие шаблоны из приложения eCommerce:

## Create Store Survey:

Type	<input type="text" value="Random Poll"/>
Group Name	<input type="text" value="testSurveyGroup"/>
Survey	<input type="text" value="Ask about BIMINFO"/>
Product ID	<input type="text"/>
Category ID	<input type="text"/>
FromDate	<input type="text"/> <input type="button" value="..."/>
ThruDate	<input type="text"/> <input type="button" value="..."/>
Survey Template Path	<input type="text" value="/templates/survey/minisurvey.ftl"/>
Result Template Path	<input type="text" value="/applications/ecommerce/templat"/>
Sequence	<input type="text" value="1"/>
	<input type="button" value="Add"/>

Рис. 6.22. Обязательные для заполнения поля при связывании опросов с определенным складом в приложении Catalog (фрагмент того же интерфейса с рис. 6.21)

- ◆ /applications/ecommerce/templates/survey/minisurvey.ftl;
- ◆ /applications/ecommerce/templates/survey/miniresult.ftl.

При подготовке интерфейса на любой созданный интерактивный опрос, связанный с текущим логическим Web-сайтом, можно сослаться по значению атрибута SurveyId и включать его в состав страницы. Обычно включение интерактивного опроса в состав интерфейса производится внутри FTL-шаблона. Причем принято включать опрос, случайно выбранный из нескольких опросов, входящих в заданную группу.

Для упрощения подготовки описания интерактивного запроса на языке HTML, выбранного случайным образом из заданной группы, используются служебные Java-классы:

- ◆ org.ofbiz.product.store.ProductStoreWorker;
- ◆ org.ofbiz.content.survey.SurveyWrapper.

```

<#assign groupName = page.randomSurveyGroup?if_exists>
<#if groupName?has_content>
  <#assign
randomSurvey=Static["org.ofbiz.product.store.ProductStoreWorker"].getRandomSurvey
Wrapper(request, "testSurveyGroup")?if_exists>
</#if>
<#if randomSurvey?has_content>
  <table border=0 width='100%' cellspacing='0' cellpadding='0' class='boxoutside'>
    <tr>
      <td width='100%'>
        <table width='100%' border='0' cellspacing='0' cellpadding='0' class='boxtop'>
          <tr> <td valign=middle align=center>
            <div class="boxhead">${randomSurvey.getSurveyName()?if_exists}</div>
          </td> </tr>
        </table>
      </td> </tr>
    <tr>
      <td width='100%'>
        <table width='100%' border='0' cellspacing='0' cellpadding='0' class='boxbottom'>
          <tr>
            <td>
              <table border="0" cellpadding="1">
                <tr> <td>
                  <form method="post" action="@ofbizUrl"/minipoll<#if
requestAttributes._CURRENT_VIEW_?exists>
/${requestAttributes._CURRENT_VIEW_}</#if></@ofbizUrl" style="margin: 0;">
                    ${randomSurvey.render().toString()}
                  </form>
                </td> </tr>
              </table>
            </td> </tr>
          </table>
        </td> </tr>
      </table>
    </td> </tr>
  </table>
<br>
</#if>

```

**Рис. 6.23.** Использование возможностей FreeMarker для генерации фрагмента с опросом

На рис. 6.23 приведен пример использования этих классов для включения в состав HTML-страницы случайного опроса, входящего в группу «testSurveyGroup».

Дополнительно в системе OFBIZ реализована служба createSurveyResponse, выполняющая всю внутреннюю работу по проверке правильности введенных ответов, сохранению результатов опроса, по ведению статистики и т.п.

### 6.7.3. Применение CMS при создании интерфейса системы *biminfo*

Безусловно, что для реализации деталей интерфейса нашего приложения *biminfo* стоит воспользоваться описанными ранее возможностями системы управления содержанием. В первую очередь это касается реализации функции интерактивных опросов и новостей — мы пользуемся фрагментами интерфейса других Web-приложений и копируем похожие по функциональности FTL-шаблоны из приложения *ecommerce*:

```
%OFBIZ_HOME%\applications\ecommerce\webapp\content\minipoll.ftl;  
%OFBIZ_HOME%\applications\ecommerce\webapp\content\factoids.ftl,
```

а потом включим их в состав экрана *rightscreen*. Таким образом, подготовка фрагмента интерфейса со случайным интерактивным опросом в нашем приложении будет осуществляться в файле *minipoll.ftl*, а новости будут подготавливаться в файле *factoids.ftl*. Для отображения списка и детального содержимого учебных курсов воспользуемся возможностью включения экземпляров содержания, подготовленных заранее в приложении CMS. Причем можно реализовать вывод списка курсов с учетом текущих параметров локализации. Это нам позволит выводить на экран пользователя только те курсы, которые читаются на его предпочтительном языке.

Разработаем логическую структуру взаимосвязи различных экземпляров содержания, определяющих наборы наших курсов. Корневым содержанием-владельцем будет единственный экземпляр содержания типа «PUBLISH\_LINK (SUBSITE)» (*WEB\_SITE\_PUB\_PT*) с оговоренным значением параметра *contentId* (пусть это будет значение «BIM\_COURSES»). Этот экземпляр будет владеть подчиненными экземплярами типа «DOCUMENT». При создании подчиненных экземпляров мы должны будем обязательно определять содержимое поля *Description* и *localeString*. Каждый подчиненный экземпляр содержания должен иметь ссылку на определенный экземпляр информационного ресурса, в котором и располагается реальное описание одного учебного курса. В нашем учебном примере такое описание

```

<screen name="left">
  <section>
    <actions>
      <entity-condition entity-name="ContentAssocViewFrom" use-cache="false"
        list-name="coursesList">
        <condition-list combine="and">
          <condition-expr field-name="caContentIdTo" operator="in"
value="BIM_COURSES"/>
          <condition-expr field-name="caContentAssocTypeId"
            operator="equals" value="DESCRIPTION"/>
          <condition-expr field-name="caThruDate" operator="equals" value=""/>
        </condition-list>
        <order-by field-name="caFromDate DESC"/>
      </entity-condition>
    </actions>
    <widgets>
      <container style="column-left">
        <platform-specific>
          <html>
            <html-template
location="component://biminfo/webapp/biminfo/includes/leftside.ftl"/>
          </html>
        </platform-specific>
        <iterate-section list-name="coursesList" entry-name="course"
view-size="6" paginate="true" paginate-
target="Course?">
          <section>
            <widgets>
              <section>
                <condition>
                  <if-compare-field field-name="course_locales"
operator="contains"
to-field-name="course.localeString"
type="String"/>
                </condition>
                <widgets>
                  <container>
                    <label text="{course.description}"/>
                  </container>
                  <container>
                    <link text="View Description..."
target="showCourseInfo?contentid={course.contentId}" style="tabButton"/>
                  </container>
                </widgets>
              </section>
            </widgets>
          </section>
        </iterate-section>
      </container>
    </widgets>
  </section>
</screen>

```

**Рис. 6.24.** Фрагмент описания экрана с определением элементов содержания, выдаваемых пользователю в зависимости от локализации

```

<screen name="showcourse">
  <section>
    <actions>
      <set field="page.appTabButtonItem" value="Main"/>
      <set field="page.headerItem" value="Course Description"/>
      <set field="contentId" from-field="requestParameters.contentId" default-
value="default_course"/>
    </actions>
    <widgets>
      <decorator-screen name="main-decorator">
        <decorator-section name="body">
          <container style="left">
            <content content-id="{contentId}" xml-escape="false"/>
          </container>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>

```

**Рис. 6.25.** Фрагмент описания экрана «showcourse»

будет храниться в информационном ресурсе типа LongText или Image. В ходе подготовки описания экрана мы должны будем извлечь из ЦУЭД элемент данных, соответствующий корневому содержимому, а затем составить список из его подчиненных экземпляров, отфильтровав по типу атрибута localeString. Такой фрагмент описания представлен на рис. 6.24.

В экране «showcourse» происходит включение описания выбранного пользователем курса (рис. 6.25).

#### *6.7.4. Загрузка данных в систему OFBIZ из внешних файлов*

Чтобы не вводить вручную много тестовых данных, определяющих некоторую вымышленную информацию по курсам, можно воспользоваться возможностью импорта информации в базу данных системы OFBIZ из внешних XML-файлов. Для загрузки внешних данных в систему biminfo нужно использовать файл из каталога

`%OFBIZ_HOME%\specialized\biminfo\data\ import_data.xml.`

Загрузить данные из XML-файла можно, находясь в Web-приложении webtools системы OFBIZ (ссылка «XML Data Import»).



К сожалению, сами двоичные данные изображений не экспортируются (по причине использования данных типа blob), поэтому файл gd.gif в основном экране CMS (рис. 6.17) загружается вручную.

### 6.7.5. Конечный результат

В результате проведенной реализации после перезагрузки системы OFBIZ и экспорта данных наш прототип будет выглядеть следующим образом (см. рис. 6.24).

Доступ к корневой странице нашего Web-приложения происходит по адресу<sup>30</sup>:

<http://localhost:8080/biminfo/>

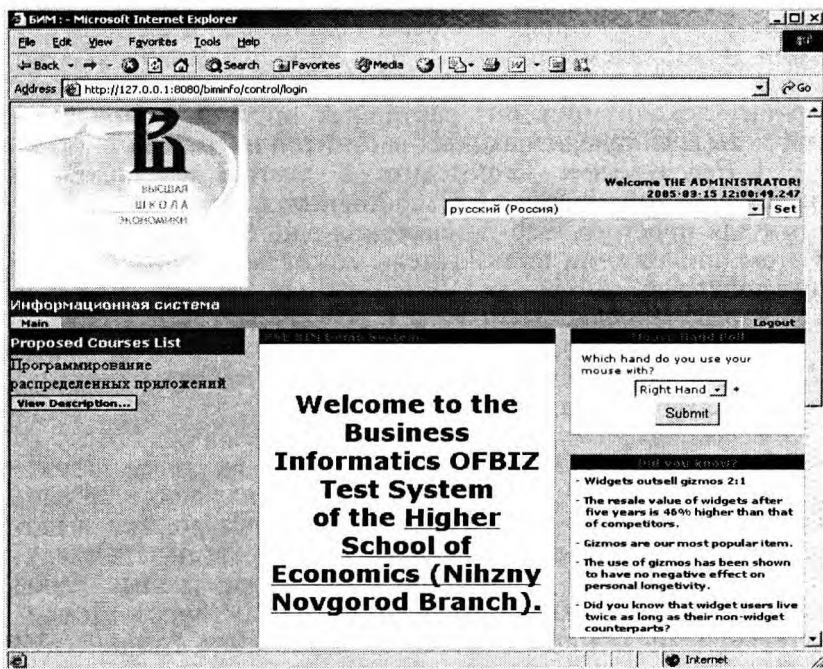


Рис. 6.26. Законченный вид приложения biminfo

<sup>30</sup> Здесь предполагается, что OFBIZ запущен на локальной машине.

## **6.8. Контрольные задания к главе 6**

1. *Групповое (состав группы произволен, размер группы 7—8 чел. До конца локализовать Web-приложение biminfo — везде заменить текстовые константы на подстановку значений переменных из файла свойств:*

**`BimUiLabels*.properties.`**

2. *Персональное.* Каждому студенту сделать так, чтобы в его локальной версии OFBIZ цвет фона в стиле, используемом для области ввода имени и пароля приложения biminfo, был определен в виде последовательности ASCII-кодов первых букв фамилии студента в латинской транскрипции. Подсказка: определите, проанализировав файл header.ftl, в каком файле каскадных стилевых таблиц (css) определяются стили HTML-элементов.

3. *Групповое.* Предложить, дать описание и реализовать вариант локализации интерактивных опросов в приложении и выдаче локализованных вариантов новостей.

4. *Персональное.* Определите в диаграмме, подобной представленной на рис. 6.1, состояния и переходы между ними для простого Web-приложения электронной торговли. В этом приложении пользователь может выполнять следующие действия:

- ◆ *регистрация;*
- ◆ *просмотр перечня товаров;*
- ◆ *просмотр подробной информации по товару;*
- ◆ *выбор товара;*
- ◆ *просмотр состояния выполнения заказа;*
- ◆ *подтверждение получения товара — закрытие заказа.*

5. *Персональное:* Расширить функциональность приложения biminfo — вместе с описанием курса должна выводиться фотография преподавателя. Подсказка: начать следует с реорганизации принципов взаимосвязи разных типов содержания, определяющих характеристики курса. Дополнительная информация: добавлять и изменять графические изображения в составе информационного ресурса можно только на закладке CMS. Если создается совершенно новое содержимое, то порядок следующий:

а) создать экземпляр Содержимого (в области Content ввести ContentId и ENTER);

б) создать экземпляр связи содержимого и информационного ресурса (в области ContentAssoc ввести ContentIdTo и ENTER);

с) создать экземпляр информационного ресурса (в области DataResource ввести DataResourceId, DataResourceTypeId, загрузить файл с картинкой и UPLOAD).

На закладке DataResource можно только просматривать изображения. При попытке загрузить — выдается ошибка.

*б. Групповое задание (доп. бонус).* Создать в форме бизнес-процесса описание действий для администратора, позволяющее удалять все связи между информационными ресурсами и содержанием, удалять экземпляры информационного ресурса и содержания из приложения CMS или Webtools.

---

## **Глава 7**

### **Принципы использования и реализации служб в системе OFBIZ**

Изученные возможности динамической генерации HTML-страниц с помощью языка OFBIZ-Widget или FTL-шаблонов позволяют определять несложные алгоритмы по обработке данных прямо в контексте страницы. Однако для сложных алгоритмов, которые требуют доступа к многочисленным данным или производят длительные вычисления, на странице нет места. В таком случае основная функциональность информационной системы должна быть реализована с помощью механизма служб OFBIZ. В этой главе мы рассмотрим назначение, архитектуру и основные методы использования центра управления службами (ЦУС) системы OFBIZ, познакомимся с правилами использования и реализации служб на языке Java.

#### ***7.1. Основное назначение и возможности Центра управления службами***

Понятие центра управления службами (ЦУС) является одним из важнейших составляющих архитектуры OFBIZ. ЦУС предоставляет программистам унифицированный механизм создания и применения совместно используемых (shareable), повторно используемых (reusable) и распределяемых (distributable) компонентов, реализующих прикладные алгоритмы. Примерами таких прикладных алгоритмов, выполненных в форме независимых компонентов, могут являться алгоритмы начисления налогов и расчетов цены в зависимости от характеристик покупателя, поиск товаров по сложным критериям, численные расчеты с помощью

внешних программ и многое другое. В системе OFBIZ подобные компоненты принято называть службами. Именно службы составляют функциональную основу любой реальной информационной системы, создаваемой на базе OFBIZ и обладают следующими важными архитектурными свойствами:

1) методы реализации и вызова службы универсальны и не зависят от особенностей Web-технологий;

2) в службе полностью определяется интерфейс вызова (число и тип входных и выходных формальных параметров, обязательные и необязательные параметры), такой интерфейс в системе OFBIZ обычно называют *моделью службы* (Service Model);

3) при каждом вызове службы производится проверка соответствия числа и типа фактических параметров формальным параметрам;

4) клиент службы (внешняя программа или модуль в системе OFBIZ) может вызывать на исполнение службу, не зная реальное место ее расположения и работы;

5) клиент службы для подготовки параметров и вызова службы использует унифицированные методы, которые совершенно не зависят от способа ее реализации.

С точки зрения разработчика, выполняющего проектирование и реализацию алгоритмов на основе служб, нужно указать на следующие детали.

1. В системе OFBIZ поддерживаются различные способы реализации служб (Java-классы, скрипты на языке интерпретирующего типа, специальные проблемно-ориентированные языки<sup>31</sup>, определения бизнес-процессов на языке XPDЛ, удаленные Web-службы на внешних серверах и т.д.). За вызов всех служб, реализованных похожим образом, отвечает один компонент — *исполнитель служб* (Service Engine).

2. Определение интерфейса службы, а также связь с конкретной реализацией и исполнителем служб происходит в текстовом конфигурационном XML-файле. Разработчик

---

<sup>31</sup> В таких языках набор ключевых слов и синтаксис выражений максимально соответствуют сущности решаемых задач в определенной предметной области, поэтому таким языкам легко обучаются специалисты. В качестве такого языка в приложении приведено описание языка Mini-Language.

служба может в файле конфигурации дать разрешение на ее использование любыми приложениями в составе системы OFBIZ, либо ограничить область видимости службы рамками всего одного приложения.

3. Вызов службы может быть синхронным и асинхронным — при синхронном вызове обращающийся к службе клиент приостанавливает свою работу до завершения выполнения службы. При асинхронном вызове клиент не дожидается завершения. Нужно отметить, что одну и ту же службу можно вызывать как в синхронном, так и асинхронном режиме.

4. Службы могут вызываться автоматически по определенному расписанию.

5. Службы могут обладать свойством «экспортируемости» (exportable). В таком случае вызов службы разрешен для различных удаленных программ-клиентов, создаваемых независимыми разработчиками. Для вызова экспортируемой службы клиенты могут использовать различные стандартные объектно ориентированные протоколы удаленного доступа (SOAP, RMI, IIOP, EJP Session Bean).

6. Службы обладают способностью вызова других служб. Это позволяет просто связывать воедино отдельные и небольшие по функциональности службы в законченные сложные процессы. Причем одни и те же службы могут использоваться в различных процессах — это заметно повышает их коэффициент повторного использования.

7. Службы легко могут быть использованы для создания Web-приложений. Обычно в этом случае к службам происходит доступ из так называемых Web-событий<sup>32</sup> (Web Events), активизирующихся в процессе обработки определенного запроса, приходящего от удаленного Web-клиента.

8. Поверх служб, реализующих прикладные алгоритмы и определяемых своей моделью, в системе OFBIZ существует несколько дополнительных типов служб. Их исполь-

---

<sup>32</sup> Web-событие, или Web-Event — чаще всего это Java-класс, полный путь к которому указывается в под-элементе event элемента request-map в конфигурационном файле controller.xml. Событие вызывается на разных этапах обработки запроса Web-клиента, возвращая определенную текстовую строку, и по значению этой строки принимается решение о подготовке того или иного представления.

зование упрощает программирование реакции на определенные события в системе, взаимодействие со стандартными источниками данных (например, почтовыми серверами), группировку служб и т.п. В настоящий момент реализованы такие дополнительные типы служб<sup>33</sup>:

а) ECA — Event-Condition-Action. Службы этого типа реализованы как определение в форме XML условий запуска действий при наступлении какого-то события. В качестве события может выступать удачное или ошибочное завершение какой-либо другой службы, обновление данных и т.п. У каждого события имеется набор характеристик, которые используются для проверки логического условия. Если логическое условие истинно, то наступает реакция на событие. Она заключается в вызове какой-либо другой службы, создании нового события и т.п.;

б) MCA — Message-Condition-Action. Службы этого типа во многом похожи на ECA-службы и применяются для автоматизации обработки почтовых сообщений. MCA-служба также определяется с помощью XML и задает правило автоматического вызова другой службы в случае получения почтового сообщения, у которого содержимое полей (адресат, тема, дата и т.п.) удовлетворяет установленному логическому условию. Система OFBIZ позволяет сконфигурировать несколько различных источников получения почтовых сообщений (разные сервера, пользователи, пароли, частота опроса, протокол и т.п.) и автоматически преобразует текстовое почтовое сообщение в набор полей ассоциативного массива для передачи в службу обычным образом;

в) Group-Services. В соответствии с названием эти службы определяют правила одновременного вызова сразу нескольких служб по какому-либо способу (последовательно, параллельно и т.п.).

Все перечисленные характеристики служб позволяют разработчикам законченных решений на базе системы OFBIZ создавать множество независимых программных компонент, а затем связывать их в сложные распределенные алгоритмы управления и обработки данных.

---

<sup>33</sup> Подробности по дополнительным типам служб системы OFBIZ вынесены в приложение.

## 7.2. Архитектура ЦУС

Общие принципы архитектуры ЦУС и ее главнейшие составные части могут быть изучены по рис. 7.1. На этом рисунке прямоугольниками обозначены отдельные Java-классы или целые программы, стрелками — вызовы методов и пути передачи данных.

*Диспетчер служб (ServiceDispatcher)* обрабатывает каждый запрос на вызов служб, перенаправляя его в требуемый *исполнитель служб (ServiceEngine)*, который реально производит вызов службы. В приложениях OFBIZ доступ к диспетчеру служб осуществляется опосредованно, через интерфейс *локального диспетчера (LocalDispatcher Interface)*. Его содержание определено в `org.ofbiz.service.LocalDispatcher`. Интерфейс локального диспетчера позволяет использовать различные способы вызова служб (синхронные, асинхронные, с тайм-аутом, с уведомлением клиента о завершении вызова, без уведомления и т.п.), проверять статус выполнения и т.п. Каждый локальный диспетчер имеет уникальное

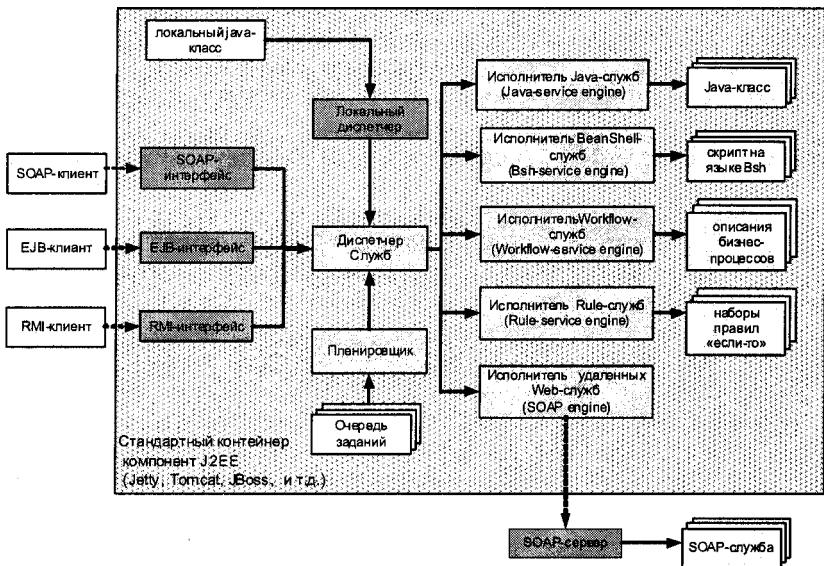


Рис. 7.1. Общая структура ЦУС



имя и содержит свой собственный список определений служб. У приложения может быть несколько экземпляров локального диспетчера, соответствующих одному и тому же диспетчеру служб.

В процессе создания экземпляра локального диспетчера формируется соответствующий *контекст вызова службы* (DispatchContext), который передается в диспетчер службы и этот же контекст помещается в состав аргументов при каждом вызове службы. Он содержит ссылки на файлы определения служб, используемый загрузчик классов, определения специфичных атрибутов и другую информацию, которая предоставляет диспетчеру служб доступ к модели службы, а службе позволяет определить, в каком контексте и для какого клиента требуется выполнять алгоритм.

*Планировщик* (JobScheduler) является многопоточным компонентом системы OFBIZ, который управляет автоматическим вызовом различных служб по определенному расписанию. Для автоматического вызова службы создается так называемое *задание* (Job), в котором сохраняется вся необходимая для вызова службы информация: входные аргументы, экземпляр диспетчера служб и пр. Когда наступает время выполнения задания, планировщик создает отдельный поток, в котором и выполняется служба.

### 7.3. Правила определения модели служб

Модель службы (т.е. все значимые характеристики службы, необходимые для ее вызова) определяется в текстовом XML-файле определения службы (Service Definition File). Существуют глобальные файлы определения службы, которые используются всеми диспетчерами, и локальные файлы определения службы, относящиеся только к определенному диспетчеру.

Структура файла определения службы описана в виде XML-схемы и находится в файле

`%OFBIZ_HOME%\framework\service\dtd\services.xsd`.

В виде UML-диаграммы эта структура изображена на рис. 7.2.

Большая часть описания службы размещается в элементе Service. Смысл его основных атрибутов определен в следующей табл. 7.1.

Кроме того, при описании каждой службы необходимо явно перечислить имена и типы всех входных и выходных параметров. Это делается с помощью последовательности вложенных под-элементов attribute (табл. 7.2).

На рис. 7.3 приведен фрагмент файла описания, в котором определяются характеристики службы с именем «userLogin».

За выполнение этой службы несет ответственность исполнитель, знающий как запускать java-классы (значение атрибута «engine» — java). Таким образом, атрибут location указывает на определенный java-класс, в котором и реализована служба. Из описания службы можно увидеть, что

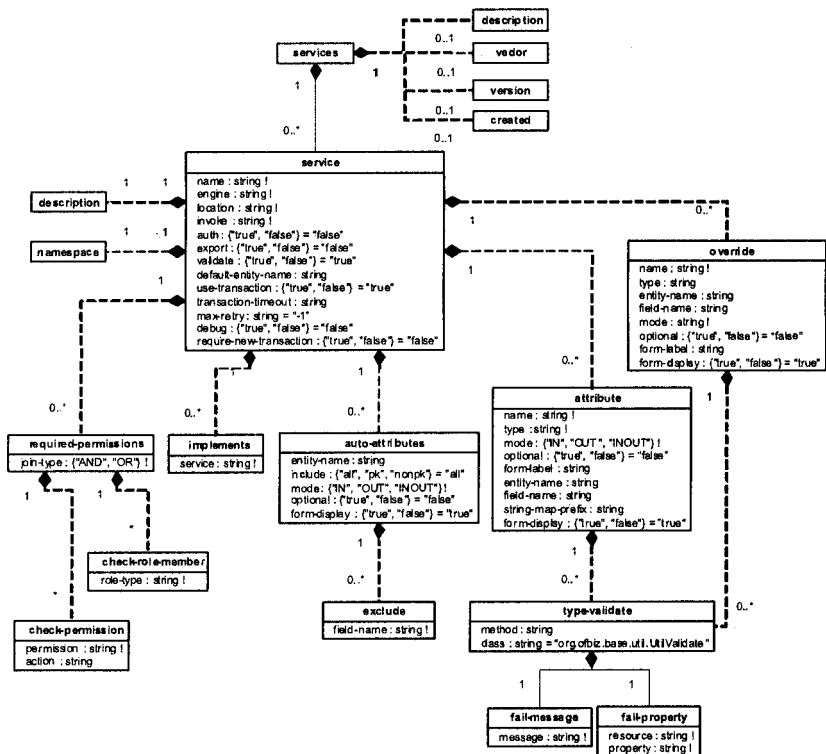


Рис.7.2. Взаимосвязь элементов в файле определения служб

## Описание атрибутов элемента Service

Имя атрибута	Требуется обязательно?	Назначение
name	Да	уникальное имя службы
engine	Да	<p>Тип определенного исполнителя служб. В системе OFBIZ реализованы следующие типы исполнителей (перечислены в конфигурационном файле serviceengine.xml)</p> <ol style="list-style-type: none"> <li>1. bsh</li> <li>2. simple</li> <li>3. java</li> <li>4. interface</li> <li>5. group</li> <li>6. http</li> <li>7. jacl</li> <li>8. javascript</li> <li>9. jms</li> <li>10. jpython</li> <li>11. route</li> <li>12. rmi</li> <li>13. soap</li> <li>14. workflow</li> </ol>
location	Да	определение местоположения службы. Формат зависит от типа исполнителя служб. Например, для исполнителя java-классов — это полное имя класса
invoke	Да	имя метода службы
auth	Нет	Требуется ли для вызова службы авторизация
export	Нет	Является ли данная служба «экспортируемой»
validate	Нет	Проверять ли правильность количества и типов атрибутов при вызове службы
default-entity-name	Нет	
use-transaction	Нет	
max-retry	Нет	
debug	Нет	
require-new-transaction	Нет	

Описание атрибутов элемента `attribute`

Имя атрибута	Требуется обязательно?	Назначение
<code>name</code>	Да	Имя атрибута
<code>type</code>	Да	Тип атрибута ( <code>String</code> , <code>java.util.Date</code> , и т.п.)
<code>mode</code>	Да	Способ использования атрибута (IN — входной, OUT — выходной, INOUT — входной/выходной)
<code>optional</code>	Нет	Является ли данный параметр необязательным
<code>form-label</code>	Нет	
<code>entity-name</code>	Нет	
<code>field-name</code>	Нет	
<code>string-map-prefix</code>	Нет	
<code>string-list-suffix</code>	Нет	
<code>form-display</code>	Нет	

```

<service name="userLogin" engine="java"
location="org.ofbiz.securityext.login.LoginServices" invoke="userLogin">
  <description>Used to Automatically Authenticate a username/password; create a
UserLogin object</description>
  <attribute name="login.username"           type="String"           mode="IN"/>
  <attribute name="login.password"          type="String"           mode="IN"/>
  <attribute name="visitId"                 type="String"           mode="IN"
optional="true"/>
  <attribute name="isServiceAuth"           type="Boolean"          mode="IN"
optional="true"/>
  <attribute name="userLogin"               type="org.ofbiz.entity.GenericValue"
mode="OUT"/>
  <attribute name="userLoginSession"        type="java.util.Map"    mode="OUT"
optional="true"/>
</service>

```

Рис. 7.3. Пример описания службы

при ее вызове ожидается два обязательных входных параметра (атрибут `mode` со значением IN): `login.username` и `login.password`.

При вызове службы проверяется наличие этих параметров и соответствие их типов. В случае несоответствия типов или отсутствия этих параметров вызов службы не происхо-

```

<service name="testSvc"
  engine="java"
  export="true"
  validate="false"
  require-new-transaction="true"
  location="org.ofbiz.common.CommonServices"
  invoke="testService"
>
<description>Test service</description>
  <attribute name="message"      type="String"      mode="IN" optional="true"/>
  <attribute name="resp"        type="String"        mode="OUT"/>
</service>

```

**Рис. 7.4.** Пример описания службы testSvc

дит, и генерируется ошибка. Кроме обязательных входных параметров могут быть использованы и необязательные входные параметры. У таких параметров в модели службы атрибут `mode` имеет значение `IN`, а атрибут `optional` — значение `true`.

После вызова службы ЦУС проверяет на наличие и правильность типов обязательных выходных параметров (атрибут `mode` со значением `OUT`). Причем, если в результате работы службы в качестве выходного параметра передается не указанный в описании, то генерируется ошибка.

## **7.4. Правила вызова служб из Java-классов**

Вызов служб чаще всего происходит из `java`-классов или в описании экранов на языке `OFBIZ-Widget`. Рассмотрим, как реализовать вызов службы из производного `Java`-класса, не связанного напрямую с `Web`-приложениями (например, вызов службы из другой службы).

Способы вызова служб из `Java`-классов мы изучим на примере простой службы (рис. 7.4.), описание которой находится в файле:

```
%OFBIZ_HOME%/framework/common/servicedef/services_test.xml.
```

Правила вызова достаточно просты. Все операции по вызову службы производятся через локального диспетчера. В компонентах, не связанных непосредственно с `Web`-приложениями, разработчику требуется явно создать экземпляр класса `GenericDispatcher` таким образом:

```
GenericDelegator delegator = GenericDelegator.getGenericDelegator("default");  
LocalDispatcher dispatcher = new GenericDispatcher("UniqueName", delegator);
```

После того, как создан экземпляр интерфейса `LocalDispatcher`, его методы можно использовать для вызова любой определенной в системе OFBIZ службы. Обычно каждая служба требует при вызове наличие определенных входных параметров. Для их хранения и передачи используется ассоциативный массив (`Map`), в который нужно последовательно поместить все обязательные входные параметры в виде пар «имя параметра»—«значение параметра». Посмотрите, например, на рис. 7.5, где проводится подготовка обязательных входных параметров для службы «testScv» из рис. 7.4 с последующим синхронным вызовом этой службы и распечаткой значения выходного параметра.

Вместо непосредственного синхронного или асинхронного вызова службы разработчик может назначить ее автоматическое исполнение на определенное время. Для этого используется механизм планировщика (`JobScheduler`). На рис. 7.6 показан пример назначения однократного автоматического вызова той же службы «testScv», а на рис. 7.7 происходит назначение серии автоматических вызовов этой службы через каждые пять секунд.

## ***7.5. Правила реализации служб на языке Java***

Теперь рассмотрим на примере все той же простой службы «testScv», как реализуются службы системы OFBIZ в виде `java`-классов. Исходный код службы «testScv» располагается в файле:

```
%OFBIZ_HOME%/framework/common/org/offbiz/common/CommonServices.java.
```

На рис. 7.8 приведен фрагмент `Java`-файла с ее реализацией.

Из рис. 7.8 видно, что служба должна быть реализована как статический метод `Java`-класса, с именем, соответствующим значению параметра «invoke» элемента `service` из файла описания службы. Этот статический метод должен принимать на вход два параметра:

- ◆ `DispatchContext dctx` — контекст вызова службы;
- ◆ `Map context` — ассоциативный массив с набором входных параметров.

```

Map context = UtilMisc.toMap("message", "This is a test.");
Map result = null;
try
{
    result = dispatcher.runSync("testSvc", context);
}
catch (GenericServiceException e)
{
    e.printStackTrace();
}
if (result != null)
    System.out.println("Result from service: " + (String) result.get("resp"));

```

**Рис. 7.5.** Пример вызова службы

```

// This example will schedule a job to run now.
Map context = UtilMisc.toMap("message", "This is a test.");
try
{
    long startTime = (new Date()).getTime();
    dispatcher.schedule("testSvc", context, startTime);
}
catch (GenericServiceException e)
{
    e.printStackTrace();
}

```

**Рис. 7.6.** Пример назначения службы на однократное автоматическое выполнение

```

// This example will schedule a service to run now and repeat once every 5 seconds a total of 10times
Map context = UtilMisc.toMap("message", "This is a test.");
try
{
    long startTime = (new Date()).getTime();
    int frequency = RecurrenceRule.SECONDLY;
    int interval = 5;
    int count = 10;
    dispatcher.schedule("testSvc", context, startTime, frequency, interval, count);
}
catch (GenericServiceException e)
{
    e.printStackTrace();
}

```

**Рис. 7.7.** Пример назначения службы на многократное автоматическое выполнение

```

public class CommonServices {

    public final static String module = CommonServices.class.getName();
    ...
    /**

    * Generic Test Service

    * @param dctx The DispatchContext that this service is operating in
    * @param context Map containing the input parameters
    * @return Map with the result of the service, the output parameters
    */
    public static Map testService(DispatchContext dctx, Map context) {
        Map response = ServiceUtil.returnSuccess();

        if (context.size() > 0) {
            Iterator i = context.keySet().iterator();

            while (i.hasNext()) {
                Object cKey = i.next();
                Object value = context.get(cKey);

                System.out.println("--- SVC-CONTEXT: " + cKey + " => " + value);
            }
        }
        if (!context.containsKey("message")) {
            response.put("resp", "no message found");
        } else {
            System.out.println("----SERVICE TEST---- : " + (String) context.get("message"));
            response.put("resp", "service done");
        }

        System.out.println("---- SVC: " + dctx.getName() + " ----");
        return response;
    }
    ...
}

```

**Рис. 7.8.** Пример реализации службы

В качестве возвращаемого результата также используется ассоциативный массив (в нашем случае — response), в который помещаются значения выходных параметров. Отметим, что в реализации службы можно использовать статический метод returnSuccess класса org.ofbiz.service.ServiceUtil. Он возвращает готовый для размещения выходных параметров ассоциативный массив, в котором уже



```

<#- display the error messages ->
<#if requestAttributes.errorMessageList?has_content>
  <div class="errorMessage">The following errors occurred:</div><br>
</ul>
  <#list requestAttributes.errorMessageList as errorMsg>
    <li class="errorMessage">${errorMsg}</li>
  </#list>
</ul>
</#if>
<#if requestAttributes.eventMessageList?has_content>
  <div class="eventMessage">The following occurred:</div><br>
</ul>
  <#list requestAttributes.eventMessageList as eventMsg>
    <li class="eventMessage">${eventMsg}</li>
  </#list>
</ul>
</#if>

```

**Рис. 7.8.** Пример FTL-шаблона для вывода информации о причинах неудачного выполнения службы

```

<services>
...
  <service name="calcTaxInterface" engine="interface" location="" invoke="">
    <description>Tax Calc Service Interface</description>
    <attribute name="productStoreId" type="String" mode="IN"/>
    <attribute name="itemProductList" type="java.util.List" mode="IN"/>
    <attribute name="itemAmountList" type="java.util.List" mode="IN"/>
    <attribute name="itemPriceList" type="java.util.List" mode="IN"/>
    <attribute name="itemShippingList" type="java.util.List" mode="IN" optional="true"/>
    <attribute name="orderShippingAmount" type="Double" mode="IN" optional="true"/>
    <attribute name="shippingAddress" type="org.ofbiz.entity.GenericValue" mode="IN" optional="true"/>
    <attribute name="orderAdjustments" type="java.util.List" mode="OUT"/>
    <attribute name="itemAdjustments" type="java.util.List" mode="OUT"/>
  </service>
...
  <service name="calcTax" engine="java"
    location="org.ofbiz.order.order.OrderServices" invoke="simpleTaxCalc">
    <description>Simple Calc Tax Service</description>
    <implements service="calcTaxInterface"/>
  </service>
...
</services>

```

**Рис. 7.10.** Определение службы calcTax

находится стандартный выходной параметр с именем «responseMessage» и значением «success». Наличие этого значения в списке возвращенных параметров сигнализирует диспетчеру об успешном завершении работы службы. Разработчик может также использовать другие варианты завершения работы службы: «error» и «fail».

В этом случае требуется использовать следующие статические методы класса ServiceUtil:

- ◆ Map returnError(String errorMessage);
- ◆ Map returnFailure(String errorMessage).

Система OFBIZ будет формировать список с именем errorMessageList, содержимое которого можно отобразить на странице с помощью средств Freemarker (рис. 7.9):

### *7.5.1. Специфика вызова служб в ходе обработки Web-запроса (controller.xml)*

После предварительного знакомства с различными аспектами определения, вызова и реализации служб в системе OFBIZ можно изучить особенности вызова служб из Web-приложений. В них интерфейс локального диспетчера доступен через объект ServletContext, который сохраняется в объекте Session. Объект Session передается в соответствующий класс-событие Event в процессе обработки Web-запроса, поэтому самостоятельно создавать экземпляр локального диспетчера не нужно.

Для пояснения взаимодействия различных составляющих системы OFBIZ в ходе реализации Web-приложения рассмотрим один сквозной пример создания и применения службы в Web-приложении. Для нашего примера подойдет Java-служба «calcTax», в которой реализован алгоритм вычисления налога на товар, в зависимости от географического местоположения покупателя и других параметров заказа. Служба «calcTax» определяется в XML-файле следующим образом (рис. 7.10):

Соответствующий java-класс с реализацией службы показан на рис. 7.11. На этом рисунке наиболее значимые части выделены жирным шрифтом.

Служба используется в Web-приложении Order и должна быть вызвана при обработке запроса calcTax. Согласно

```

1. package org.ofbiz.order.order;
2.
3.     import java.util.List;
4.     import java.util.Map;
5.
6.     import org.ofbiz.base.util.UtilMisc;
7.     import org.ofbiz.entity.GenericDelegator;
8.     import org.ofbiz.entity.GenericValue;
9.     import org.ofbiz.service.DispatchContext;
10.
11. /** Simple tax calc service. */
12. public static Map simpleTaxCalc(DispatchContext dctx, Map context)
13. {
14.     GenericDelegator delegator = dctx.getDelegator();
15.     String productStoreId = (String) context.get("productStoreId");
16.     List itemProductList = (List) context.get("itemProductList");
17.     List itemAmountList = (List) context.get("itemAmountList");
18.     List itemPriceList = (List) context.get("itemPriceList");
19.     List itemShippingList = (List) context.get("itemShippingList");
20.     Double orderShippingAmount = (Double) context.get("orderShippingAmount");
21.     GenericValue shippingAddress = (GenericValue) context.get("shippingAddress");
22.     GenericValue userLogin = (GenericValue) context.get("userLogin");
23.
24.     //Simple Tax Calc only uses the state from the address and the SalesTaxLookup.
25.
26.     String countryCode = null;
27.     String stateCode = null;
28.
29.     if (shippingAddress != null)
30.     {
31.         countryCode = shippingAddress.getString("countryGeold");
32.         stateCode = shippingAddress.getString("stateProvinceGeold");
33.     }
34.
35.     // Setup the return lists.
36.     List orderAdjustments = new ArrayList();
37.     List itemAdjustments = new ArrayList();
38.
39.     // Loop through the products; get the taxCategory; and lookup each in the cache.
40.     for (int i = 0; i < itemProductList.size(); i++)
41.     {
42.         GenericValue product = (GenericValue) itemProductList.get(i);
43.         Double itemAmount = (Double) itemAmountList.get(i);
44.         Double itemPrice = (Double) itemPriceList.get(i);
45.         Double shippingAmount = (Double) itemShippingList.get(i);
46.         List taxList = null;
47.         if (shippingAddress != null)
48.         {
49.             taxList = getTaxAmount(delegator,
50.                                     product,
51.                                     productStoreId,
52.                                     countryCode,
53.                                     stateCode,
54.                                     itemPrice.doubleValue(),
55.                                     itemAmount.doubleValue(),
56.                                     shippingAmount.doubleValue());

```

Рис. 7.11. Реализация службы calcTax

```

57.         itemAdjustments.add(taxList);
58.     }
59.     }
60.     if (orderShippingAmount.doubleValue() > 0)
61.     {
62.         List taxList = getTaxAmount(delegator,
63.                                     null,
64.                                     productStoreId,
65.                                     countryCode,
66.                                     stateCode,
67.                                     0.00,
68.                                     0.00,
69.                                     orderShippingAmount.doubleValue());
70.     }
71.     orderAdjustments.addAll(taxList);
72. }
73. }
74. Map result = UtilMisc.toMap("orderAdjustments", orderAdjustments,
75.                             "itemAdjustments", itemAdjustments
76.                             );
77.
78. return result;
79. }

```

**Рис. 7.11.** Реализация службы calcTax (окончание)

```

<request-map uri="calcTax">
  <event type="java" path="org.ofbiz.order.shoppingcart.CheckOutEvents"
  invoke="calcTax"/>
  <response name="success" type="request" value="validatePaymentMethods"/>
  <response name="error" type="request" value="checkouterror"/>
</request-map>

```

**Рис. 7.12.** Фрагмент файла controller.xml  
с указанием вызова Web-события

```

package org.ofbiz.order.shoppingcart;
...
public class CheckOutEvents
{
...
public static String calcTax(HttpServletRequest request,
                             HttpServletResponse response)
{
    try
    {
        calcTax(request);
    }
    catch (GeneralException e)
    {

```

**Рис. 7.13.** Пример вызова службы из Web-события

```

        request.setAttribute("_ERROR_MESSAGE_", e.getMessage());
        return "error";
    }
    return "success";
}
//----- Invoke the taxCalc

private static void calcTax(HttpServletRequest request) throws GeneralException
{
    LocalDispatcher dispatcher = (LocalDispatcher) request.getAttribute("dispatcher");
    ShoppingCart cart = (ShoppingCart) request.getSession().getAttribute("shoppingCart");
    //Calculate and add the tax adjustments for first shipping Group
    ShoppingCart.CartShipInfo csi = cart.getShipInfo(0);
    int totalItems = csi.shipItemInfo.size();
    List product = new ArrayList(totalItems);
    List amount = new ArrayList(totalItems);
    List price = new ArrayList(totalItems);
    List shipAmt = new ArrayList(totalItems);
    Iterator iter = csi.shipItemInfo.keySet().iterator();
    for (int i = 0; i < totalItems; j++)
    (
        ShoppingCartItem cartItem = (ShoppingCartItem) csi.shipItemInfo.get(j);
        ShoppingCart.CartShipInfo.CartShipItemInfo itemInfo =
        (ShoppingCart.CartShipInfo.CartShipItemInfo) csi.shipItemInfo.get(cartItem);
        product.add(i, cartItem.getProduct());
        amount.add(i, new Double(cartItem.getItemSubTotal(itemInfo.quantity)));
        price.add(i, new Double(cartItem.getBasePrice()));
        shipAmt.add(j, new Double(0.00));
    )
    Double shipAmount = new Double(csi.shipEstimate);
    shipAddress = cart.getShippingAddress(shipGroup);
    Map serviceContext = UtilMisc.toMap("productStoreId", productStoreId);
    serviceContext.put("itemProductList", product);
    serviceContext.put("itemAmountList", amount);
    serviceContext.put("itemPriceList", price);
    serviceContext.put("itemShippingList", shipAmt);
    serviceContext.put("orderShippingAmount", shipAmount);
    serviceContext.put("shippingAddress", shipAddress);

    Map serviceResult = null;

    try
    (
        serviceResult = dispatcher.runSync(«calcTax», serviceContext);
    )
}
catch (GenericServiceException e)
(

```

**Рис. 7.13.** Пример вызова службы из Web-события (продолжение)

```

Debug.logError(e, module);
throw new GeneralException
    ("Problem occurred in tax service (" + e.getMessage() + ")", e);
}
if (ServiceUtil.isError(serviceResult))
{
    throw new GeneralException(ServiceUtil.getErrorMessagе(serviceResult));
}
}
...
}

```

Рис. 7.13. Пример вызова службы из Web-события (окончание)

требованиям системы OFBIZ в файле `controller.xml` необходимо указать при определении этого запроса особый `java`-класс — Web-событие (`Event`). Внутри этого класса и будет происходить вызов службы «`calcTax`». В нашем случае таким классом-событием является `org.of-biz.order.shoppingcart.Check OutEvents` (рис. 7.12).

Фрагмент этого класса, где происходит подготовка входных параметров, вызов службы и обработка результата выполнения, показан на рис. 7.13.

### 7.5.2. Инициализация служб на старте компонента

В составе конфигурации компонента системы OFBIZ может быть создано несколько файлов определения служб с произвольными именами. Чтобы на этапе инициализации компонента об этих файлах стало известно, системе требуется перечислить их в главном конфигурационном файле компонента `ofbiz-component.xml`.

На рис. 7.14 представлен главный конфигурационный файл компонента `common`

`%OFBIZ_HOME%\framework\common\ofbiz-component.xml`),

в котором жирным цветом выделены строчки с перечислением используемых файлов определения служб.

Из этого рисунка видно, что разработчик обычно создает отдельный файл определения служб для каждой из четырех различных категорий служб системы OFBIZ:

- 1) `model` — простые службы;
- 2) `group` — групповые службы;

```

<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="common" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/ofbiz-
component.xsd">
  <resource-loader name="main" type="component"/>
  <classpath type="jar" location="build/lib/**"/>
  <classpath type="dir" location="config"/>
  <classpath type="dir" location="script"/>
  <entity-resource type="model" reader-name="main" loader="main"
location="entitydef/entitymodel.xml"/>
  <entity-resource type="group" reader-name="main" loader="main"
location="entitydef/entitygroup.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/CommonTypeData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/CountryCodeData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/CurrencyData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/GeoData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/GeoData_BR.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/GeoData_US.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/LanguageData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/UnitData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/PeriodData.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_test.xml"/>
  <service-resource type="group" loader="main"
location="servicedef/groups_test.xml"/>
  <service-resource type="eca" loader="main"
location="servicedef/secas_test.xml"/>
  <service-resource type="mca" loader="main"
location="servicedef/smcas_test.xml"/>
</ofbiz-component>

```

**Рис. 7.14.** Указание файлов определения служб в конфигурационном файле компонента ofbiz-component.xml

- 3) еса — службы-реакции на события;
- 4) мса — службы-реакции на почтовые сообщения.

В оставшейся части главы основное внимание сосредоточено на объяснении правил создания и использования

служб, относящихся к категории model и соответствующего формата файла определения служб. Краткая информация о службах категорий group, esa и tsa, их особенностях и отличиях от model дана в приложении.

### 7.5.3. Интерактивный запуск служб из Web-приложения webtools

Администратор системы OFBIZ имеет удобную возможность назначить запуск любой службы из Web-приложения webtools (рис. 7.15).

В этом приложении администратор имеет возможность указать имя вызываемой службы, ввести ее входные параметры, установить время и периодичность запуска службы в планировщике (рис. 7.16).

The screenshot shows a web browser window titled "OFBiz: Core Web Tools - Microsoft Internet Explorer". The address bar shows "https://127.0.0.1:8443/webtools/control/scheduleJob". The page header includes the OFBiz logo and the text "OPEN FOR BUSINESS OFBiz.org". A navigation menu contains items like "Accounting", "BIN INFO", "Blog", "Catalog", "Control", "Facility", "Manufacturing", "Marketing", "Order", "Party", "Supply", "WebTools", and "WorkPart". The main content area is titled "Framework Web Tools" and contains a form for "Schedule A Job".

**Schedule A Job**  
Step 1: Service & Recurrence Information

Service:

Pool Name:

Start Date/Time:

Finish Date/Time:

Frequency:

Interval:  (for use with frequency)

Count:  (number of time the job will run, use -1 for no limit i.e. forever)

Max Retry:  (number of time the job will retry on error; use -1 for no limit or leave empty for service default)

(c) 2004 The Open For Business Project - www.ofbiz.org  
Powered by OFBiz

Рис. 7.15. Web-приложение для интерактивного запуска служб



File Edit View Favorites Tools Help

Address: https://127.0.0.1:8443/webtools/control/joblist

**OPEN FOR BUSINESS**  
OFBiz.org

Welcome THE ADMINISTRATOR!  
2005-04-27 15:48:32.783  
English (United States) Set

Accounting | **FIN INFO** | Blog | Catalog | Content | Facility | Manufactures | Marketing | Order | Party | Share | WebTools | **WorkFlow**

Framework Web Tools

Scheduled Jobs  
[Refresh] [Thread List] [Service Log] [Schedule Job]

Job	Pool	Run Time	Start Time	Service	Finish Time	
Run Auto-Reorders	pool	2805-04-28 04:89:88.8		runShoppingListAutoReorder		[Cancel Job]
Run Auto-Reorders	pool	2805-04-28 04:89:90.8		runShoppingListAutoReorder		[Cancel Job]
Order Auto-Cancel	pool	2005-04-28 03:08:00.0		autoCancelOrderItems		[Cancel Job]
Order Auto-Cancel	pool	2005-04-28 03:08:00.0		autoCancelOrderItems		[Cancel Job]
Re-Try Failed Auths	pool	2005-04-28 01:00:00.0		retryFailedAuths		[Cancel Job]
Re-Try Failed Auths	pool	2005-04-28 01:00:00.0		retryFailedAuths		[Cancel Job]
Clear EntitySyncRemove Info	pool	2005-04-28 09:00:00.0		cleanSyncRemoveInfo		[Cancel Job]
Purge Old Store Auto-Entered Promos	pool	2005-04-28 00:00:00.0		purgeOldStoreAutoPromos		[Cancel Job]
Purge Old Store Auto-Entered Promos	pool	2005-04-28 00:00:00.0		purgeOldStoreAutoPromos		[Cancel Job]
BackOrder Notification	pool	2005-04-28 00:00:08.8		checkInventoryAvailability		[Cancel Job]
Clear EntitySyncRemove Info	pool	2005-04-28 09:00:00.0		cleanSyncRemoveInfo		[Cancel Job]
Purge Old Jobs	pool	2005-04-28 08:00:00.0		purgeOldJobs		[Cancel Job]
Purge Old Jobs	pool	2005-04-28 08:00:00.0		purgeOldJobs		[Cancel Job]
BackOrder Notification	pool	2005-04-28 00:00:00.0		checkInventoryAvailability		[Cancel Job]
Run Auto-Reorders	pool	2805-04-27 04:00:00.0	2005-04-27 11:56:54.525	runShoppingListAutoReorder	2005-04-27 11:57:14.283	

Internet

Рис. 7.16. Перечень заданий, находящихся под управлением планировщика

## 7.6. Создание службы для Web-приложения biminfo

Все полученные знания возможно использовать для реализации законченного Web-приложения с развитыми возможностями. Примем упрощенную модель обучения в ВШЭ, в которой студент самостоятельно может выбирать набор дисциплин для обучения в каждом модуле. Отдельная дисциплина оценивается деканатом определенным количеством баллов (кредитов), и на оценку общей успеваемости студента влияет сумма баллов всех выбранных в модуле дисциплин (чем больше сумма — тем больше оценка). Кроме того, каждая дисциплина субъективно оценивается студентом с точки зрения сложности изучения. Набор дисциплин достаточно большой, поэтому сделать правильный выбор путем последовательного перебора вариантов сложно. Для помощи студентам в процессе планирования

CatalogCompany Name: Page Title Edit Content - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Media Print Mail

Address https://127.0.0.1:8443/content/control/EditContentAttribute?contentId=BIM\_COURSE\_3

**OPEN FOR BUSINESS**  
OFBiz.org

Accounting BIM INFO Blog Catalog Content Faculty Manufacturing Marketing Order Entry CRM

**Content Manager Application**

View Edit Delete Add

Find	Content	Association	Role	Purpose	Attribute
Content Id	Attr Name	Attr Value			
BIM_COURSE_3	Cost	20	Update	[Delete]	

Content Id: BIM\_COURSE\_3

Attr Name: \_\_\_\_\_

Attr Value: \_\_\_\_\_

Add

Рис. 7.17. Создание нового атрибута с именем Cost для содержания

индивидуального учебного плана на модуль необходимо с помощью системы OFBIZ реализовать функциональность «интеллектуального» планировщика. Входными данными для него служат:

- ◆ множество предлагаемых к изучению дисциплин вместе с их характеристиками: количеством баллов и субъективной оценкой сложности, даваемой дисциплине самим студентом;

- ◆ значение максимальной суммарной трудоемкости, превышать которое построенное расписание не должно.

Планировщик на основании этой информации должен построить оптимальное расписание, — т.е. выбрать такое подмножество дисциплин, на котором достигается максимальное значение суммы баллов, а суммарное значение трудоемкости меньше или равно максимальной суммарной трудоемкости, заданной ранее студентом. Класс таких задач хорошо изучен в математической теории и носит название «задачи о рюкзаке». Существует эффективный алгоритм решения задачи о рюкзаке, которым мы можем в ходе реализации воспользоваться.

Прежде всего для решения этой задачи нам потребуется в закладке Content Web-приложения webtools дополнить каждый экземпляр содержания, описывающего учебную дисциплину, новым атрибутом с названием Cost. В этом атрибуте будет храниться количество баллов (кредитов), назначенных дисциплине деканатом (рис. 7.17). Дополнительно потребуется заменить у курсов локализацию en на en\_US.

Реализацию графического интерфейса задачи мы проведем в составе модуля biminfo, добавив в его графический интерфейс дополнительный пункт меню Schedule Advisor и соответствующие экраны с формами для ввода исходных данных и представления результата (рис. 7.18, 7.19).

Правила переходов между различными формами и экранами определяются следующей диаграммой (рис. 7.20).

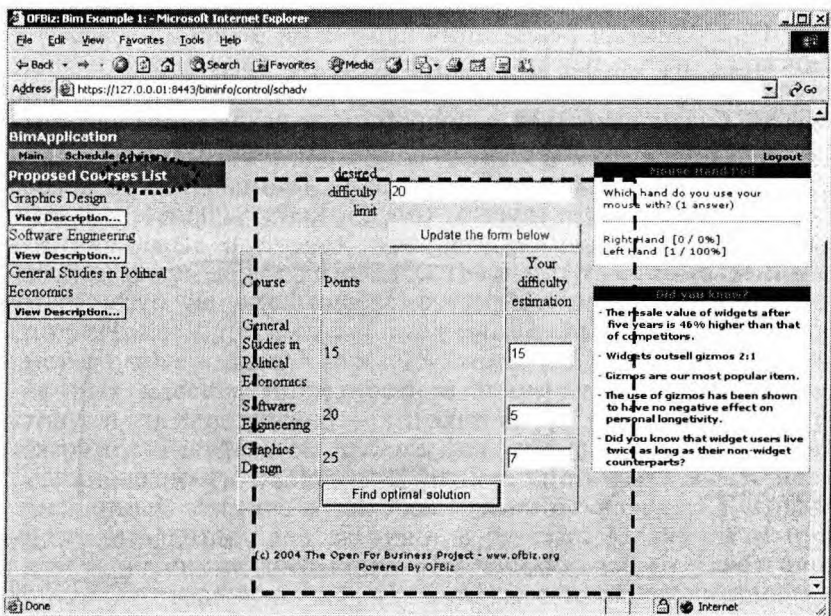


Рис. 7.18. Экран для ввода исходных данных, необходимых планировщику

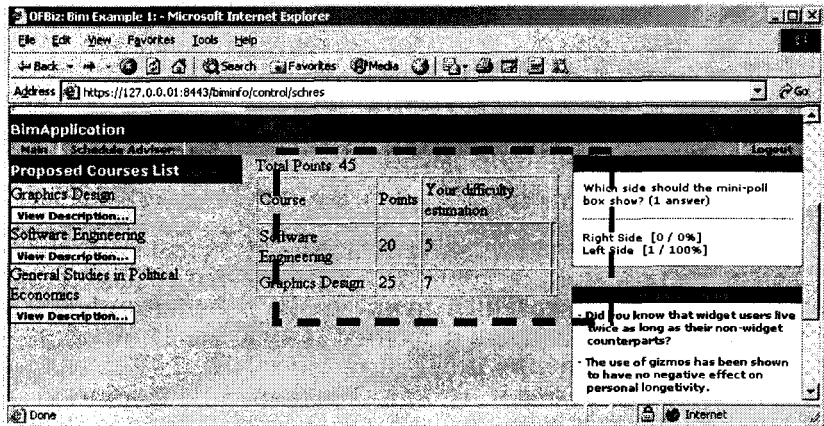


Рис. 7.19. Экран с выводом результатов работы планировщика

Программная реализация алгоритма решения «задачи о рюкзаке» взята из другой системы и воплощена в виде отдельного Java-класса `org.hse.bim.schedule.KnapsackSolver.java`. Мы не будем вдаваться в подробности реализации алгоритма решения задачи о рюкзаке в общем виде. Скажем лишь,

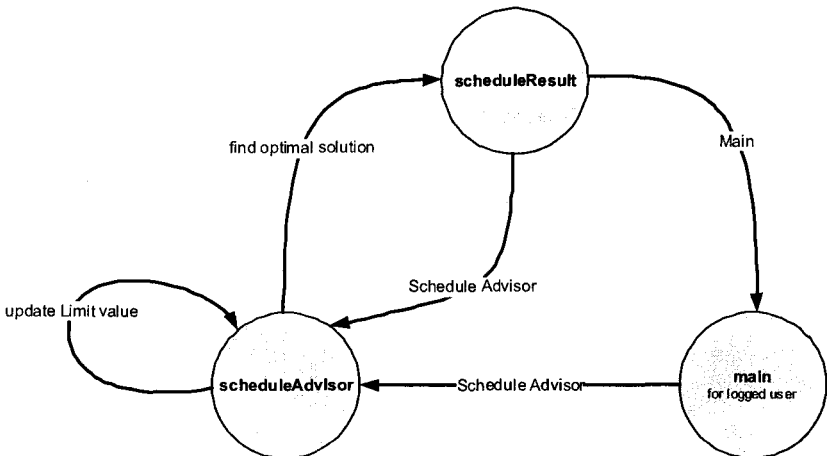


Рис. 7.20. Правила переходов между экранами

```

<?xml version="1.0" encoding="UTF-8"?>
<services xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/services.xsd">
  <description>Example Services</description>
  <vendor>OFBiz</vendor>
  <version>1.0</version>
  <service name="solveKnapsack" auth="false" engine="java" invoke="solveKnapsack"
location="org.hse.bim.schedule.ScheduleServices">
    <description>Service that solves an ordinary knapsack problem</description>
    <attribute name="numberOfItems" type="Integer" mode="IN" optional="false"/>
    <attribute name="storageLimit" type="Integer" mode="IN" optional="false"/>
    <attribute name="goodnessList" type="java.util.List" mode="IN" optional="false"/>
    <attribute name="storageList" type="java.util.List" mode="IN" optional="false"/>
    <attribute name="results" type="java.util.List" mode="OUT" optional="false"/>
    <attribute name="value" type="Integer" mode="OUT" optional="false"/>
  </service>
  <service name="contentlist" engine="bsh" location="prepContentList.bsh" invoke="">
    <description>Composes a string "a|b|c" from the list</description>
    <attribute name="contentList" type="java.util.List" mode="IN" optional="true"/>
    <attribute name="contentListPiped" type="String" mode="OUT"/>
  </service>
</services>

```

**Рис. 7.21.** Определение интерфейса службы в файле  
 %OFBIZ\_HOME%\specialized\biminfo\servicedef\services.xml

что в общей постановке задачи имеется набор предметов с заданным значением полезности (goodness) и занимаемого места (storage). Необходимо определить такое решение (загружаемые в рюкзак предметы), которое имеет максимальную суммарную полезность, а его суммарно занимаемое место не превышает некоторого ограничения. В используемой программной реализации на вход статического метода solve класса KnapsackSolver передаются два целочисленных массива (полезность и занимаемое место предметов) и целое число — ограничение на максимальную загрузку. Результатом работы алгоритма является суммарная полезность найденной оптимальной загрузки и целочисленный массив, в котором единицы находятся в позициях выбранных для загрузки в рюкзак предметов и нули для оставшихся предметов. Полный исходный текст алгоритма оптимальной укладки в рюкзак приведен в приложении. Как очевидно из предыдущих примеров определения и использования служб, в системе OFBIZ для передачи данных между различными компонентами используются слу-

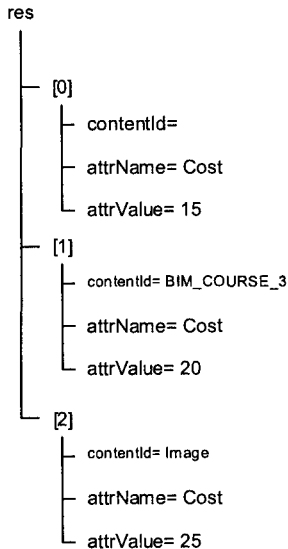
```

<screen name="scheduleAdvisor">
  <section>
    <actions>
      <set field="page.appTabButtonItem" value="Main"/>
      <set field="page.headerItem" value="Course Description"/>

      <set field="a.a" from-field="locale" default-value="en_US"/>
      <entity-condition entity-name="ContentAssocViewFrom" use-cache="false" list-
name="coursesIds">
        <condition-list combine="and">
          <condition-expr field-name="caContentIdTo" operator="in"
value="BIM_COURSES"/>
          <condition-expr field-name="caContentAssocTypeId" operator="equals"
value="DESCRIPTION"/>
          <condition-expr field-name="caThruDate" operator="equals" value=""/>
          <condition-expr field-name="localeString" operator="equals" env-
name="a.a"/>
        </condition-list>
        <order-by field-name="caFromDate DESC"/>
      </entity-condition>
      <service service-name="contentlist" result-map-name="result">
        <field-map env-name="coursesIds" field-name="contentList"/>
      </service>
      <entity-condition entity-name="ContentAttribute" use-cache="false" list-
name="res">
        <condition-list combine="and">
          <condition-expr field-name="attrName" operator="equals" value="Cost"/>
          <condition-expr field-name="contentId" operator="in" env-
name="result.contentListPiped"/>
        </condition-list>
      </entity-condition>
    </actions>
    <widgets>
      <decorator-screen name="main-decorator">
        <decorator-section name="body">
          <container style="column-right">
            <include-form name="difficulty_level"
location="component://biminfo/widget/forms.xml"/>
          </container>
          <container style="main" >
            <include-form name="courses"
location="component://biminfo/widget/forms.xml"/>
          </container>
        </decorator-section>
      </decorator-screen>
    </widgets>
  </section>
</screen>

```

**Рис. 7.22.** Определение экрана «scheduleAdvisor» в файле %OFBIZ\_HOME%\specialized\biminfo\widget\MainScreens.xml



**Рис. 7.23.** Фрагмент модели данных OFBIZ, определяющий содержимое таблицы из рис. 7.18

жебные классы-контейнеры языка Java (например, `java.util.Map`, `java.util.List`). Содержимым таких классов-контейнеров не могут являться атомарные типы данных (`int`, `double`, `boolean`), используемые в алгоритме Knapsack Solver. Поэтому нам необходимо создать отдельную службу системы OFBIZ, в которой будет выполняться необходимое преобразование данных и обращение к методу `solve` класса `KnapsackSolver`. Описание модели этой службы выглядит так, как показано на рис. 7.21.

Из определения модели видно, что служба совершенно не учитывает специфики механизмов кодирования и передачи данных из форм HTML-страницы на сервер OFBIZ. Поэтому в принципе вызов этой службы может осуществляться различными внешними клиентами с помощью разных протоколов удаленного взаимодействия. Это является большим преимуществом использования механизма служб системы OFBIZ. Но в рамках нашей конкретной задачи

единственным клиентом службы является Web-браузер, из которого исходные данные передаются службе. Поэтому нам потребуется прежде всего создать определение новых экранов и форм на языке OFBIZ-Widget. Формат представления данных, передаваемых из формы на HTML-странице в систему OFBIZ, специфичен и отличается от ожидаемых службой. Поэтому для организации связи Web-браузера и службы нам будет нужно реализовать дополнительный Java-класс для конвертации данных из специфичного HTML-формата в набор входных аргументов службы. Рассмотрим вначале структуру нового экрана «scheduleAdvisor», на котором пользователь может определить желаемый уровень суммарной трудоемкости выбранных дисциплин и индивидуальную сложность каждой дисциплины (рис. 7.22).

В определении экрана заслуживает внимания секция actions. В этой секции выполняются операции по формированию фрагмента модели данных системы OFBIZ. Этот фрагмент состоит из списка res, в каждом элементе которого хранятся значения трех полей: идентификатор курса, название атрибута, значение атрибута (рис. 7.23).

Поля заполняются значениями из соответствующих экземпляров сущностей ContentAttribute по следующему алгоритму. Сначала происходит выборка экземпляров сущностей Content, у которых локализация соответствует текущей выбранной пользователем, а идентификатор родительского содержания равен BIM\_COURSES (т.е. мы выбираем экземпляры содержания, которые определяют учебные курсы, читаемые на выбранном пользователем языке). На основе выбранных экземпляров сущностей Content формируется список courseIds, состоящий из идентификаторов экземпляров содержания (атрибут ContentId). Затем этот список с помощью службы contentlist<sup>34</sup> преобразуется в простую текстовую строку с именем result.contentListPiped, в которой каждый идентификатор отделяется друг от друга символом «|» (такая строка с точки зрения системы OFBIZ выражает множество значений). Наконец, производится поиск экземпляров сущностей ContentAttribute, у которых значение атрибута ContentId содержится в этом множестве.

---

<sup>34</sup> Определение этой службы можно найти в файле:

%OFBIZ\_HOME%\specialized\biminfo\servicedef\servi-ces.xml.



```

<form name="difficulty_level" type="single" default-map-name="difficulty" >
  <actions>
    <set field="difficulty.level" from-field="requestParameters.level" global="true"/>
  </actions>
  <field name="level" title="desired difficulty limit" title-area-style="main" >
    <text maxlength="25" size="25"/>
  </field>
  <field name="submitButton" title="Update the form below" >
    <submit button-type="button" />
  </field>
</form>

```

**Рис. 7.24.** Определение формы «» в файле  
 %OFBIZ\_HOME%\specialized\biminfo\widget\forms.xml

```

<form name="courses" target="schres" type="multi" use-row-submit="false" list-
name="res">
  <field name="level" map-name="difficulty">
    <hidden/>
  </field>
  <field name="contentId" title="Course">
    <display-entity entity-name="Content"/>
  </field>
  <field name="attrValue" title="Points">
    <display/>
  </field>
  <field name="estimation" title="Your difficulty estimation">
    <text maxlength="5" size="5"/>
  </field>
  <field name="submitButton" title="Find optimal solution">
    <submit button-type="button"/>
  </field>
</form>

```

**Рис. 7.25.** Определение формы «courses» в файле  
 %OFBIZ\_HOME%\specialized\biminfo\widget\forms.xml

Экран `scheduleAdvisor` использует шаблон декоратора, поэтому по сути в секции `widgets` нам потребуется в определении экрана включить лишь две новые формы. Структура формы «`difficulty_level`» для ввода суммарной трудоемкости проста, и мы приводим ее определение без дополнительных комментариев на рис. 7.24, 7.25.

Вторая форма «`courses`» для определения трудоемкости каждой отдельной дисциплины более сложная (рис. 7.25). Для нее выбран тип «`multi`», что определяет особый вид

представления информации на экране браузера и формат передаваемых на сервер данных. Наша форма типа «multi» выглядит как таблица, состоящая из трех столбцов и произвольного количества строк. Количество строк и конкретное содержимое ячеек таблицы определяются значением полей списка с именем «ges» из модели данных OFBIZ.

Все поля формы, кроме поля с именем level, получают значения из соответствующих полей этого списка. Поле формы с именем level получает значение из ассоциативного массива difficulty. Причем значение этого поля формы во всех строчках таблицы одинаково. При передаче данных из формы типа multi на сторону сервера информация кодируется в составе одного единственного HTTP-запроса по следующим правилам:

- ◆ содержимое запроса состоит из последовательности пар <имя параметра>=<значение параметра>, разделяемых символом ‘;’;

- ◆ каждой ячейке таблицы соответствует уникальное имя параметра HTTP-запроса в формате <имя столбца>\_o\_<номер строки>;

- ◆ кроме пользовательских данных в составе запроса передается служебный параметр с именем «\_rowCount». Значением этого параметра является число строк в таблице;

- ◆ если при определении формы на языке OFBIZ-Widget в элементе form указан атрибут «use-row-submit=true», то на экране браузера в таблице появляется дополнительный столбец для выбора строки (checkbox), а в состав HTTP-запроса для каждой строки добавляется служебный параметр «\_rowSubmit» (значения «Y» или «N») и единственный на весь запрос служебный параметр «\_rowCount=Y».

Таким образом, HTTP-запрос для таблицы с рис. 7.18 выглядит следующим образом (рис. 7.26):

Теперь рассмотрим механику обработки переданных из форм данных на стороне сервера. Из предыдущих разделов известно, что в параметре target элемента form задается символическое имя запроса. Это имя используется в файле WEB-INF/controller.xml для определения группы конкретных действий, выполняемых системой OFBIZ в ходе обработки запроса.

```
contentId_o_0=BIM_COURSES_2;attrValue_o_0=15;estimation_o_0=15;contentId_o_1=BIM_COURSES_3;attrValue_o_1=20;estimation_o_1=5; contentId_o_2=Image; attrValue_o_2=25; estimation_o_2=7;_row_count=3
```

**Рис. 7.26.** Формат HTTP-запроса при передаче данных из формы типа multi

В том случае, когда в определении формы параметр `target` отсутствует (например, в форме «`difficulty_level`» из рис. 7.24), клиенту возвращается та же страница, на которой исходно расположена форма. При этом происходит обновление модели данных OFBIZ — в нашем случае в ассоциативный массив (Map) с именем «`difficulty`» заносится новое значение с ключом «`level`».

Согласно определению формы «`courses`» (рис. 7.25), результатом обработки ее данных должна явиться передача клиенту новой страницы, сформированной по запросу «`schres`». Рассмотрим, как определяются в файле

#### WEB-INF/cont-roller.xml

действия, которые должна выполнить система OFBIZ для подготовки этой страницы (рис. 7.27).

```
<request-map uri="schres">
  <security https="true" auth="true"/>
  <event type="java" path="org.hse.bim.web.bimEvents" invoke="calcSchedule"/>
  <response name="success" type="view" value="scheduleResult"/>
</request-map>

.....

<view-map name="showCourseInfo" type="screen"
page="component://biminfo/widget/MainScreens.xml#showcourse"/>
  <view-map name="scheduleAdvisor" type="screen"
page="component://biminfo/widget/MainScreens.xml#scheduleAdvisor"/>
  <view-map name="scheduleResult" type="screen"
page="component://biminfo/widget/MainScreens.xml#scheduleResult"/>
```

**Рис. 7.27.** Определение действий по запросу «`schres`» в файле `%OFBIZ_HOME%\specialized\biminfo\webapp\biminfo\WEB-INF\controller.xml`

Прежде всего в ходе выполнения запроса должны быть обработаны данные, переданные из формы. Для этого используется элемент `event` (Web-событие). По исходной задумке авторов системы OFBIZ для обработки данных,

```

<form name="proposedCourses" type="list" list-name="resList">
  <field name="contentId" title="Course">
    <display-entity entity-name="Content"/>
  </field>
  <field name="attrValue" title="Points">
    <display/>
  </field>
  <field name="estimation" title="Your difficulty estimation">
    <display/>
  </field>
</form>

```

**Рис. 7.28.** Определение формы «proposedCourses» в файле %OFBIZ\_HOME%\specialized\biminfo\widget\forms.xml

приходящих из форм типа multi, может использоваться особый тип Web-события с типом «service-multi». При использовании события такого типа в его параметрах указывается имя службы, чья модель совпадает с определением полей формы. Служба автоматически вызывается столько раз, сколько строк передано из формы (причем, если в определении формы был указан атрибут «use-row-submit=true»,

то в службу будут передаваться данные только из тех строк, которые были выбраны пользователем).

В большинстве ситуаций такой вариант обработки данных полностью подходит для разработчика, но в нашем случае это не так — служба (solveKnapsack) должна быть вызвана ровно один раз. Поэтому нам потребуется создать специальный java-класс и использовать его в роли обычного Web-события типа java. В этом классе будет выполнена трансформация параметров HTTP-запроса из формата рис. 7.26 в структуру данных, ожидаемую службой. Этот класс будет относиться к категории WebEvent и его определенный статический метод будет вызываться сервлетом-контроллером. Таким образом, мы должны указать полное имя класса и имя метода в параметрах элемента event. Полный текст класса представлен в приложении.

В случае успешной трансформации параметров HTTP-запроса и вызова службы solveKnapsack в классе org.hse.bim.web.bimEvents контроллер производит формирование представления scheduleResult. Это представление оп-

```

<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="biminfo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/ofbiz-
component.xsd">
  <!-- define resource loaders; most common is to use the component resource loader -->
  <resource-loader name="main" type="component"/>
  <!-- load single or multiple external libraries -->
  <classpath type="jar" location="build/lib/*"/>
  <!-- place the config directory on the classpath to access configuration files -->
  <classpath type="dir" location="config"/>
  <classpath type="dir" location="script"/>
  <service-resource type="model" loader="main" location="servicedef/services.xml"/>
  <!-- web applications; will be mounted when using the embedded Jetty container -->
  <webapp name="biminfo" title="BIM INFO" server="default-server"
location="webapp/biminfo" mount-point="/biminfo"/>
</ofbiz-component>

```

**Рис. 7.29.** Обновленный конфигурационный файл компонента biminfo

ределено на языке OFBIZ-Widget и является экраном с именем scheduleResult в файле:

```
%OFBIZ_HOME%\specialized\biminfo\widget\MainScreens.xml.
```

В этот экран включается форма proposedCourses (рис. 7.28).

Мы завершили разбор нововведений в определение представлений, запросов и экранов, требующихся для реализации новой функциональности Web-приложения biminfo. Дополнительно нам потребуется слегка модифицировать содержимое конфигурационного файла (рис. 7.29) компонента:

```
%OFBIZ_HOME%\specia-lized\biminfo\ofbiz-component.xml.
```

Модификация этого файла заключается в указании группы каталогов и jar-архивов, которые будут использоваться системой OFBIZ для регистрации служб и поиска файлов в момент вызова java-классов и скриптов. Каталоги и jar-файлы, в которых будет производиться поиск нужных классов, определяются с помощью элементов classpath. Файлы с определениями служб указываются в элементе service-resource.

Для полного завершения работ над расширением функциональности компонента biminfo потребуется всего лишь откомпилировать java-классы, в которых реализована служба solveKnapsack и Web-событие calcSchedule. Проще всего

это сделать, если на вашей машине установлена программа ant — Java-аналог утилиты make. В этом случае все инструкции по компиляции файлов и упаковке их в jar-архив уже определены в файле build.xml. Поэтому нужно будет, находясь в каталоге %OFBIZ\_HOME%\specialized\biminfo, из консоли набрать команду ant. Если ant на вашей машине отсутствует, то после компиляции всех файлов в каталоге src их потребуется поместить в jar-архив<sup>35</sup> (имя произвольно), а архив скопировать в каталог:

```
%OFBIZ_HOME%\specialized\biminfo\build\lib.
```

Текст скриптовой службы contentlist, размещенной в файле:

```
%OFBIZ_HOME%\specialized\biminfo\script\prepContentList.bsh,
```

компилировать и архивировать не нужно. После всех сделанных изменений можно перезапускать систему OFBIZ и тестировать вновь созданную функциональность Web-приложения biminfo.

## 7.7. Контрольное задание к главе 7

1. Создайте службу OFBIZ и соответствующего Web-клиента, реализующих калькулятор. Начинать можно с простого варианта, когда калькулятор выглядит на экране браузера как форма с четырьмя полями (первый аргумент, второй аргумент, операция, результат). Самый сложный вариант должен максимально походить на обычный интерфейс калькулятора в ОС MS Windows.

2. H&N Forever: компания «Рога и Копыта» с оптимизмом восприняла демонстрацию возможностей системы OFBIZ и всерьез рассматривает перспективы реализации модели производства «Just-in-Time» с поддержкой Web-интерфейса. Вам потребуется реализовать демонстрационный прототип информационной системы (интерфейс и службы) для такой задачи с использованием jar-архива, в котором реализована базовая модель структуры и производства корпорации «Рога и Копыта». Созданная на основе

---

<sup>35</sup> Убедитесь, что в созданном jar-архиве каталоги начинаются с уровня org, а не src!

OFBIZ информационная система должна позволять выполнение следующих операций:

- a) регистрация пользователя в системе;
- b) задание необходимого количества продукции;
- c) проверка наличия нужных материалов на складе;
- d) производство и оповещение пользователя о произведенной продукции;
- e) отображение текущих параметров производства (структура фабрик, количество запасов и т.п.) для выделенной группы пользователей (только с правами администратора!);
- f) построение прогноза на объемы заказов продукции на основе накопленной статистики обращений и производства.

Реализацию служб нужно проводить с учетом следующих требований:

- a) обновление запасов на складе проводится через каждые пять обращений клиента;
- b) необходимо сохранять во внутренних структурах данных службы историю обращений для каждого клиента (имя, дата, количество запрошенной продукции, результат);
- c) «модуль предсказаний» необходимо выполнить в виде службы, автоматически запускающейся через определенный интервал времени и выдающей прогноз на консоль (простой `System.out.println`).

3. Создайте службу для проведения вычислений через систему Matlab. Эта служба должна иметь один обязательный входной аргумент — текст скрипта на М-языке системы Matlab и один выходной аргумент — результат вычисления.

4. Модифицируйте службу `SolveKnapsack` к «экспортируемой» форме. Самостоятельно изучите основы механизма удаленного вызова служб с помощью протокола SOAP и создайте простого удаленного клиента на языке Java, обращающегося к службе с помощью этого протокола<sup>36</sup>.

5. Создайте MCA-службу и несколько дополнительных обычных служб, которые в совокупности со службой

---

<sup>36</sup> Для выполнения этого задания рекомендуется ознакомиться с любым доступным пособием по Java-XML и SOAP.

`solveKnapsack` реализуют такую возможность: клиент посылает письмо на особый адрес — в этом письме по определенным соглашениям указаны параметры вызова службы `solveKnapsack` — с этими параметрами в системе `OFBIZ` решается задача о рюкзаке — полученное решение отправляется по почте клиенту. Для реализации отправки почты обратите внимание на службы:

**sendMail**

**и**

**sendGenericNotificationEmail,**

определенные в файле:

`%OFBIZ_HOME%\applications\content\servicedef\services.xml`.

При выполнении этого задания вам потребуется настроить параметры опроса почтовых серверов в файле:

`%OFBIZ_HOME%\base\config\ofbiz-containers.xml`.



---

## **Глава 8**

### **Принципы использования центра управления элементами данных (Entity Engine) в системе OFBIZ**

По мнению одного из основоположников теоретических основ информатики, Н. Вирта, состав любой программы может быть выражен с помощью следующей формулы:

**ПРОГРАММА = АЛГОРИТМЫ + СТРУКТУРЫ ДАННЫХ.**

Применяя эту формулу к системе OFBIZ, можно сказать, что первое слагаемое мы уже рассмотрели в гл. 7 — реализация алгоритмов в системе OFBIZ чаще всего выполняется с помощью служб. Поэтому для полного знакомства с арсеналом создания полнофункциональных программ нам остается познакомиться с механизмами хранения и обработки структур данных, реализованных в системе OFBIZ. Основу этих механизмов составляют понятия *элемент данных (Entity)* и *центр управления элементами данных (Entity Engine)* — ЦУЭД. Для изучения этих понятий полезно уже здесь кратко привести сведения, которые подробно рассматриваются в курсе «Базы данных».

#### ***8.1. Необходимые базовые сведения по современным принципам информационного моделирования и обработки данных***

Современные информационные технологии невозможно представить без специализированных программных средств обработки информации. Эти средства хранят, управляют и обрабатывают большие массивы структурирован-

ных данных, предоставляя к ним одновременный доступ многим пользователям. Основным программным средством такого вида являются системы управления базами данных (СУБД). Необходимость в разработке теоретических и прикладных вопросов создания этого особого класса программного обеспечения стала значительно ощущаться в начале 60-х гг. В этот момент недостатки традиционных подходов к долговременному хранению и обработке информации с помощью файлов начали оказывать огромное негативное влияние на весь дальнейший прогресс в развитии информационных систем общего пользования.

Традиционно разработчики прикладных программ (написанных, например, на Бейсике, Паскале С/С++, Java и т.п.) размещают нужные им данные в файлах, организуя их наиболее удобным для себя образом. При этом одни и те же данные могут иметь в разных приложениях совершенно разную организацию (разную последовательность размещения в записи, разные форматы одних и тех же полей и т.п.). Обеспечить возможность коллективного доступа к таким данным чрезвычайно трудно: например, любое изменение структуры записи файла, производимое одним из разработчиков, приводит к необходимости изменения другими разработчиками тех программ, которые используют записи этого файла. Для иллюстрации обратимся к примеру, приведенному в книге У. Девиса «Операционные системы» (М.: Мир, 1980):

*«Несколько лет назад почтовое ведомство (из лучших побуждений) пришло к решению, что все адреса должны обязательно включать почтовый индекс. Во многих вычислительных центрах это, казалось бы, незначительное изменение привело к ужасным последствиям. Добавление к адресу нового поля, содержащего шесть символов, означало необходимость внесения изменений в каждую программу, использующую данные этой задачи в соответствии с изменившейся суммарной длиной полей. Тот факт, что какой-то программе для выполнения ее функций не требуется знания почтового индекса, во внимание не принимался: если в некоторой программе содержалось обращение к новой, более длинной записи, то в такую программу вносились изменения, обеспечивающие дополнительное место в памяти.*

*В условиях автоматизированного управления централизованной базой данных все такие изменения связаны с функциями управляющей программы базы данных. Программы, не использующие значения почтового индекса, не нуждаются в модификации — в них, как и прежде, в соответствии с запросами посылаются те же элементы данных. В таких случаях внесенное изменение неощутимо. Модифицировать необходимо только те программы, которые пользуются новым элементом данных».*

С целью повышения эффективности обработки данных, использующихся совместно в различных программах, было предложено предоставить разработчикам новое средство. Таким средством стали *базы данных (БД)* — именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области. Соответственно, совокупность языковых и программных средств, предназначенных для создания, ведения и совместного применения БД многими пользователями, получила название *система управления базами данных (СУБД)*.

По сравнению с простейшими информационными системами на основе файлов, СУБД обладает рядом характерных возможностей. К ним относятся:

1) представление в явном виде описаний используемых структур хранения и методов доступа к данным (так называемые «мета-данные»). Пользователям доступ к мета-данным возможен через служебные БД, называемые словарями или системными каталогами;

2) наличие средств администрирования для модификации структуры существующих БД и создания новых. Эти средства позволяют без обращения к услугам разработчиков модифицировать и создавать новые структуры хранения, изменять параметры работы системы;

3) специализированные встроенные языковые средства для получения и модификации хранимых в СУБД данных;

4) поддержка многопользовательского доступа к данным, включая параллельные обновления одной БД многими пользователями;

5) средства уменьшения избыточности и поддержка зависимости данных;

6) средства автоматического резервного копирования и восстановления данных;

7) обеспечение повышенной надежности хранения данных на физическом уровне за счет распределения информации по разным носителям и прочих методов;

8) средства безопасности, защищающие информацию от несанкционированного доступа и модификации, а также нарушения логической непротиворечивости (целостности) данных.

Считается, что основным преимуществом СУБД является *независимость данных*. Под этим понимают независимость средств, которые используют пользователи и прикладные программисты для доступа к данным, от их реальных методов хранения в БД и алгоритмов доступа. Физические методы хранения данных составляют физический уровень представления, а пользовательские методы носят название логического уровня представления.

СУБД поддерживает на логическом уровне строго определенный набор абстрактных структур хранения и методов доступа — *модель данных*. Она выбирается таким образом, чтобы скрыть от пользователей большинство машино- и программно-зависимых алгоритмов и структур данных. Таким образом, при организации доступа к хранимой в БД информации с помощью методов логического уровня не требуется изменять прикладные программы или обучать пользователей новым навыкам работы при изменении физических методов хранения и доступа к данным. Подобный подход иллюстрируется на рис. 8.1.

На этом рисунке понятию «логический уровень представления» соответствует понятие «Инфологическая модель данных». Именно в терминах инфологической модели данных (ее еще называют концептуальной моделью) пользователи СУБД и прикладные программисты определяют большинство своих действий с данными. Поэтому в ходе создания инфологической модели одной из главных задач является обеспечение наиболее естественных для человека способов представления и обработки той информации, которую предполагается хранить в создаваемой базе данных. Важнейшую роль в решении этой задачи играет выбор правильного языка для описания модели. Обычно он имеет большое

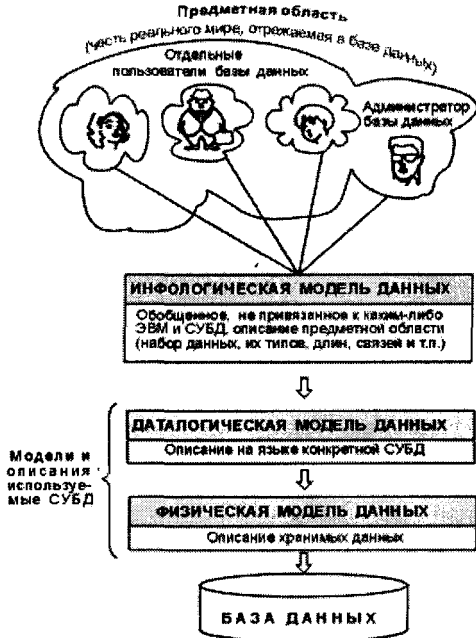


Рис. 8.1. Уровни моделей данных в СУБД

сходство с естественным языком<sup>37</sup> и не носит на себе отпечатка используемых программных технологий. В большинстве случаев язык для построения инфологических моделей включает в себя следующие конструктивные элементы.

♦ *Сущность* — любой различимый объект (объект, который мы можем отличить от другого), информацию о котором необходимо хранить и использовать в программе. Сущностями могут быть люди, места, самолеты, рейсы, вкус, цвет и т.д. Необходимо различать такие понятия, как *тип сущности* и *экземпляр сущности*. Понятие тип сущности относится к набору однородных личностей, предметов,

<sup>37</sup> Последний не может быть использован в чистом виде из-за сложности компьютерной обработки текстов и неоднозначности любого естественно-го языка.

событий или идей, выступающих как целое. Экземпляр сущности относится к конкретной вещи в наборе. Например, типом сущности может быть ГОРОД, а экземпляром — Москва, Киев и т.д.

◆ *Атрибут* — поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей (например, ЦВЕТ может быть определен для многих сущностей: СОБАКА, АВТОМОБИЛЬ, ДЫМ и т.д.). Атрибуты используются для определения того, какая информация должна быть собрана о сущности. Примерами атрибутов для сущности АВТОМОБИЛЬ являются ТИП, МАРКА, НОМЕРНОЙ ЗНАК, ЦВЕТ и т.д. Здесь также существует различие между типом и экземпляром. Тип атрибута ЦВЕТ имеет много экземпляров или значений: Красный, Синий, Банановый, Белая ночь и т.д., однако каждому экземпляру сущности присваивается только одно значение атрибута. Абсолютное различие между типами сущностей и атрибутами отсутствует. Атрибут является таковым только в связи с типом сущности. В другом контексте атрибут может выступать как самостоятельная сущность. Например, для автомобильного завода цвет — это только атрибут продукта производства, а для лакокрасочной фабрики цвет — тип сущности.

◆ *Ключ* — минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Для сущности РАСПИСАНИЕ ключом является атрибут НОМЕР\_РЕЙСА или набор: ПУНКТ\_ОТПРАВЛЕНИЯ, ВРЕМЯ\_ВЫЛЕТА и ПУНКТ\_НАЗНАЧЕНИЯ (при условии, что из пункта в пункт вылетает в каждый момент времени один самолет).

◆ *Связь* — ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако одно из основных требований к организации базы данных — это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между

ними определенные связи. А так как в реальных базах данных нередко содержатся сотни или даже тысячи сущностей, то теоретически между ними может быть установлено более миллиона связей. Наличие такого множества связей и определяет сложность инфологических моделей.

Результатом создания инфологической модели является высокоуровневое представление информационных требований, например, такое, как диаграмма «сущность-связь». Основу этой диаграммы составляет набор сущностей, который представляет или моделирует определенную совокупность сведений, специфицированную в требованиях. Сущности описаны атрибутами, позволяющими детализировать свойства сущности. Один или несколько атрибутов могут служить идентификатором для обозначения отдельных экземпляров сущности. Связи между сущностями отображают функциональные аспекты информации, представленной сущностями.

Существует несколько подходов к построению диаграмм типа «сущность-связь». Общим для всех подходов является набор из четырех основных проектных решений или шагов:

- 1) определение сущностей;
- 2) определение атрибутов сущностей;
- 3) идентификация ключевых атрибутов сущностей;
- 4) определение связей между сущностями.

Построенные диаграммы «сущность-связь» служат основой для проектирования и реализации даталогической (еще ее называют логической) модели данных, в которой определяется как сущности, атрибуты и связи реализуются средствами конкретной СУБД.

За время развития теории баз данных для использования на уровне даталогической модели было предложено около тридцати моделей данных на основе различных математических аппаратов. Однако все модели можно разделить на два класса: теоретико-множественные модели и теоретико-графовые. В теоретико-графовых моделях хранимая информация представляется в виде графа. Вершины графа — это структурированные записи, содержащие информацию о сущностях предметной области. Ребра в теоретико-графовых моделях предоставляют средства для перехо-

да от одной вершины к другой и могут рассматриваться как бинарные отношения на множестве вершин, причем на эти отношения накладывается свойство функциональности, т.е. должно быть определено, что в отношении является аргументом, а что — значением функции. Наиболее известными моделями этого класса являются иерархическая и сетевая модель данных.

*Иерархическая модель* является деревом. Каждая вершина дерева представляет определенный тип сущностей, а ребра выражают отношение «отец → сын» с некоторой конкретной семантикой.

*Сетевая модель* — модель ориентированных графов, где вершины — множество однородных сущностей, а дуги (в терминах сетевой модели они называются наборами) — ассоциации с функциональной зависимостью «владелец набора → член набора».

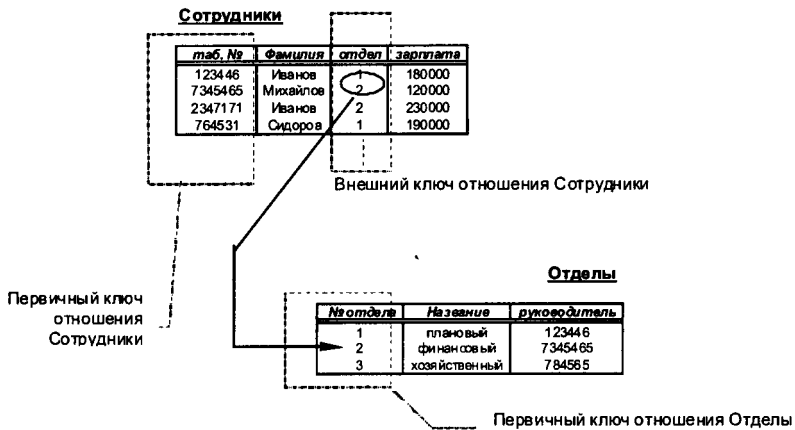
Теоретико-множественные модели представляют всю совокупность информации в виде набора множеств и предоставляют группу теоретико-множественных операций для манипулирования этими множествами. Как наиболее простая и удобная в практической работе, наибольшее распространение среди моделей этого класса получила *реляционная модель данных*<sup>38</sup>. Эта модель используется в большинстве современных СУБД.

В реляционной модели данных информация о сущностях, атрибутах и их взаимосвязях представляется с помощью двумерных структур — таблиц (отношений) с определенными именами (рис. 8.2). Таблица содержит произвольное число строк (записей) с одним и тем же фиксированным числом ячеек у всех записей. Ячейки принято называть полями. Каждый отдельный вертикальный ряд из полей всех записей одной таблицы составляет столбец таблицы. Каждый столбец имеет имя-идентификатор, уникальный в пределах своей таблицы. В любом столбце таблицы все значения полей принадлежат к определенному типу значе-

---

<sup>38</sup> Важным теоретическим результатом является полная эквивалентность выразительной мощности основных моделей данных (сетевой, иерархической, реляционной). Таким образом выбор подходящей модели обусловлен лишь предпочтениями разработчиков и пользователей СУБД, а также требованиями эффективности.





**Рис. 8.2.** Структура связанных между собой реляционных отношений

ний или, в терминах СУБД, — домену. Нестрого говоря, понятие домена эквивалентно понятию типа данных в языках программирования. В исходной реляционной модели принято использовать лишь атомарные типы значений, например, числа, строки, даты и т.п. Значения атомарных типов данных нельзя разложить на более мелкие компоненты. Составные типы данных, такие как массивы, записи

<b>AND</b>	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

<b>OR</b>	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

<b>NOT</b>	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	FALSE
UNKNOWN	UNKNOWN	UNKNOWN

**Рис. 8.3.** Таблицы истинности трехзначной логики при использовании значения NULL

или множества, в качестве типов атрибутов недопустимы. В реляционной модели поля записи независимо от своего типа могут иметь особое значение — NULL. Следует помнить, что ни численный ноль, ни пустая строка символов или бит не могут заменять значение NULL. Они являются обычными значениями, в то время как NULL указывает, что на данный момент времени значение поля неизвестно. При использовании значений NULL в логических операциях обычная двухзначная логика заменяется трехзначной введением дополнительного логического значения UNKNOWN (рис. 8.3).

Введение в реляционную модель значения NULL позволяет моделировать в БД элемент неопределенности, присущей реальному миру. Порядок записей в реляционных отношениях не определен, поэтому для их однозначной идентификации внутри таблицы выделяют множество атрибутов, совокупность значений которых уникальна для каждой записи в пределах одной таблицы. Множество таких атрибутов носит название *первичный ключ отношения* (*Primary Key*). Ограничения реляционной модели требуют, чтобы изначально и после любых модификаций данных в таблице не существовало записей с одинаковыми значениями первичного ключа. Таким образом, каждая запись может быть в любой момент однозначно определена через его значение.

Сущности предметной области моделируются в реляционной модели группой таблиц. При этом одному типу сущностей предметной области ставится в соответствие определенная таблица. Каждая отдельная запись этой таблицы содержит значения атрибутов одного экземпляра.

Логические связи сущностей предметной области определяются через связь таблиц. Для указания наличия связи между таблицами используется *внешний ключ* (*Foreign Key*). *Внешний ключ* — это множество дополнительных атрибутов в связанной таблице, совокупность значений которых является первичным ключом для другой таблицы, с которой есть связь. Используя значение внешнего ключа в качестве критерия поиска, для каждой записи из одной таблицы можно определить связанные записи из других таблиц.

Реляционная модель вводит ряд предопределенных правил на значения, которые могут иметь первичные и внешние ключи отношений. Эти правила носят название ограничений целостности и служат для сохранения логической непротиворечивости хранимой в БД информации. Например, для значений первичных ключей ограничения целостности таковы:

1) запрещается наличие в таблице записей, имеющих одинаковые значения первичных ключей;

2) ни одно из подмножеств полей, составляющих первичный ключ, не может иметь особое значение NULL.

Для внешних ключей ограничения целостности требуют, чтобы значение любого внешнего ключа соответствовало первичному ключу какой-либо существующей записи из связанной таблицы. Данное ограничение формализует наше представление о том, что связи могут быть лишь между реально существующими объектами.

Кроме предопределенных ограничений целостности, в реальных БД также могут быть описаны специфические ограничения для конкретной предметной области, такие, как ограничение множества возможных значений для атрибута и т.п. В реляционной модели для доступа к хранимой в таблицах информации было предложено несколько различных математических аппаратов. Наиболее известным является аппарат реляционной алгебры. *Реляционная алгебра* рассматривает каждое отношение как множество (в математическом смысле) записей и предоставляет две группы операторов для манипуляций с этими множествами. В результате последовательного применения операторов к исходным и промежуточным отношениям получаются новые отношения, содержащие интересующий результат. Таким образом, в отличие от традиционных файловых систем, ориентированных на построчную обработку информации, реляционная алгебра позволяет оперировать таблицами целиком.

В первую группу операторов реляционной алгебры входят стандартные теоретико-множественные операции объединения, пересечения, разности и декартова произведения множеств. Вторую группу составляют так называемые реляционные операторы. Это два унарных оператора

селекции и проекции и один бинарный оператор объединения. Оператор *селекции* позволяет выделять из отношения-аргумента записи, значение полей которых удовлетворяет определенному критерию. Такие записи составляют новое отношение. Оператор селекции записывается как  $\sigma_{\text{условие}}$  ТАБЛИЦА. Приведем пример использования этого оператора для вывода информации о сотрудниках по фамилии Иванов, используя отношения из рис. 8.2:

$\sigma_{\text{фамилия=Иванов}}$  СОТРУДНИКИ

таб. №	Фамилия	отдел	зарплата
123446	Иванов	1	180000
2347171	Иванов	2	230000

Оператор *проекции* служит для выборки из исходного отношения указанного подмножества столбцов и последующего удаления дубликатов из получившегося отношения. Запись оператора производится следующим образом:  $\pi_{\text{атрибуты}}$  ТАБЛИЦА. Используя все те же отношения рис. 8.2, получим все различные значения фамилий сотрудников:

$\pi_{\text{фамилия}}$  СОТРУДНИКИ

Фамилия
Иванов
Михайлов
Сидоров

Наиболее важным реляционным оператором является оператор *объединения* (Join) по атрибутам. Два отношения могут быть объединены, если оба имеют хотя бы один атрибут совпадающего типа. В случае возможности применения оператора объединения к двум отношениям-аргументам результатом объединения явится новое отношение. Это отношение состоит из записей, получающихся объединением каждой записи из первого отношения-аргумента с такой записью второго отношения-аргумента, что значения атрибутов, по которым происходит объединение, у этих записей совпадают. Определяется оператор объединения как

><  
атрибут1=атрибут2.

Покажем применение этого оператора для выдачи вместе с данными о сотрудниках, данных об отделах, где они работают (объединение таблиц СОТРУДНИКИ и ОТДЕЛЫ происходит по полям «отдел» и «№ отдела»):

СОТРУДНИКИ ><<sub>отдел=№отдела</sub> ОТДЕЛЫ

таб.№	Фамилия	отдел	зарплата	№ отдела	Название	руководитель
123446	Иванов	1	180000	1	плановый	123446
7345465	Михайлов	2	120000	2	финансовый	7345465
2347171	Иванов	2	230000	2	финансовый	7345465
764531	Сидоров	1	190000	1	плановый	123446

В сложных запросах различные операторы реляционной алгебры могут быть вложены друг в друга:

$\pi_{\text{фамилия, название}}$  (СОТРУДНИКИ ><<sub>отдел=№отдела</sub> ОТДЕЛЫ)

Фамилия	Название
Иванов	плановый
Михайлов	финансовый
Иванов	финансовый
Сидоров	плановый

Из приведенных примеров видно, что применение операторов реляционной алгебры в качестве средств доступа к записям позволяет пользователям полностью освободиться от ограничений физического представления и не использовать в работе машинно-зависимых конструкций языков программирования типа указателей.

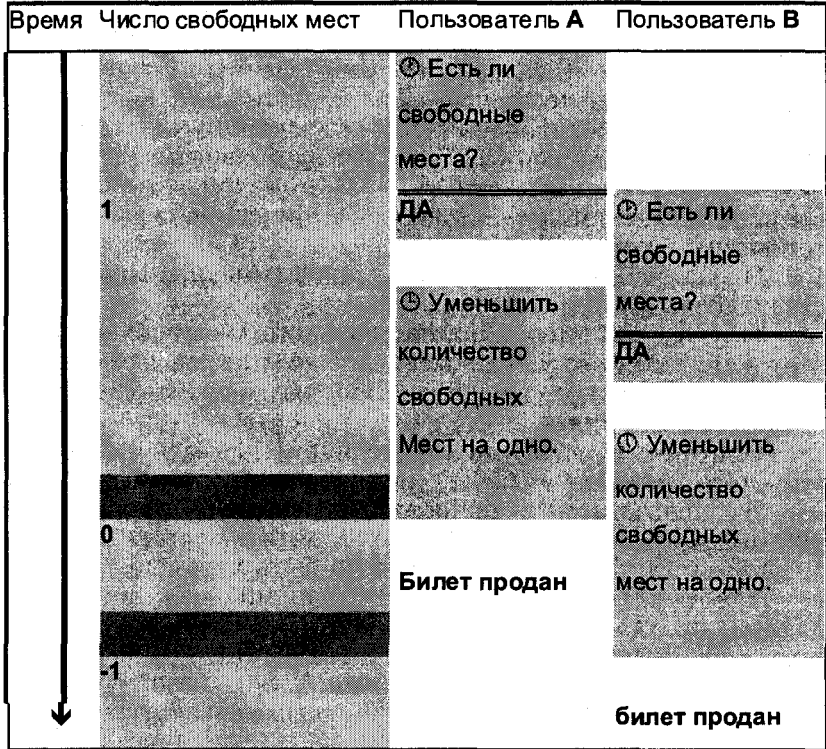
В рамках этой книги мы не будем рассматривать особенности моделей данных и специфические алгоритмы на физическом уровне. Остановимся лишь на кратком рассмотрении вопросов, связанных с защитой данных от последствий одновременной работы большого числа независимых пользователей, так как любому разработчику и пользователю информационной системы рано или поздно

придется с ними столкнуться. Многопользовательский доступ к базам данных, хранимым под управлением единого сервера БД, несомненно, обладает множеством преимуществ перед работой в локальном варианте. Однако подобный режим порождает много проблем, связанных с сохранностью и логической целостностью данных. Эти проблемы появляются в результате нарушения работоспособности сервера или в результате коллизий, возникающих при одновременном обращении к данным. Кроме того, информации угрожают намеренные или случайные опасные действия пользователей.

Содержание и объемы информации, хранимой на сервере БД, зачастую так важны, что даже кратковременное отключение сервера или утечка данных могут поставить на грань банкротства даже очень крупную фирму. Чтобы максимально обезопасить хранимую информацию, в СУБД класса предприятий применяется совокупность методов, включающая резервирование информации, восстановление информации, поддержание целостности и ограничение доступа. Комплексное использование этих методов позволяет максимально повысить безопасность данных.

*Резервирование информации* предполагает дублирование файлов данных и (или) основных физических узлов компьютера. Резервные копии файлов или аппаратного обеспечения носят названия теневого (shadow copy) или зеркальных (mirror copy). Наиболее критичным звеном в современной компьютерной технике являются различные устройства хранения информации и, в частности, как наиболее скоростной, накопитель на жестком магнитном диске. Именно эти устройства чаще всего дублируются.

Для *восстановления информации*, безвозвратно потерянной либо из-за сбоя, либо из-за ошибочной модификации БД, чаще всего применяют журнализацию изменений. *Журнализация* используется совместно с архивным копированием данных на высоконадежные устройства копирования информации. В зависимости от интенсивности обновления данных проводят ежедневное, еженедельное или квартальное архивное копирование структуры БД и данных на оптические диски, ленточные накопители и другие подобные устройства. При использовании журнализации сервер



**Рис. 8.4.** Нарушение целостности данных при одновременном доступе к ним двух пользователей

ведет протокол работы с базами данных, называемый журналом изменения данных. В нем хранится перечень всех операций над базой со времени последнего архивного копирования. Журнал может храниться в виде распечатки или файла и сохраняется особо тщательно. В случае обнаружения ошибок в данных или при фатальном сбое можно восстановить последнюю архивную копию базы и внести в нее изменения по журналу. Восстановление и изменения производятся с использованием особого программного обеспечения, входящего в состав инструментальных средств сервера БД.

Задачи *поддержания логической непротиворечивости данных* становятся очень актуальными при одновременном обновлении одной и той же базы данных множеством независимых пользователей. Для иллюстрации нарушения логической непротиворечивости приведем пример доступа многих пользователей к базе данных авиабилетов. Два независимых пользователя А и В с малой разницей во времени обращаются к БД для заказа авиабилета на один и тот же рейс (рис. 8.4).

Продажу билета клиентская программа производит по следующему алгоритму:

- 1) перед продажей билета проверить наличие мест на этот рейс. Если свободных мест нет, то отказать в продаже;
- 2) уменьшить количество свободных мест на одно;
- 3) продать билет.

Описанный алгоритм, подходящий для однопользовательского режима работы, при одновременном доступе многих пользователей приводит к неверным результатам: обоим пользователям продан билет, хотя в действительности одному из них в продаже должно быть отказано.

Чтобы избежать подобных ситуаций, для доступа к данным вводят механизм транзакций, который реализуется в виде отдельной подсистемы сервера СУБД.

*Транзакция* — группа операций над базой данных, обладающая четырьмя концептуальными свойствами (так называемые свойства ACID).

◆ *Атомарность (atomicity)*. Это свойство определяет неразделяемость операций в транзакции. Таким образом, всегда гарантируется, что полностью выполняются все операции, составляющие транзакцию, либо ни одна из операций не выполняется. Атомарность транзакции сохраняется даже при фатальных сбоях сервера БД в момент завершения части ее операций. В этом случае для восстановления состояния, в котором находилась БД на момент запуска транзакции, используется журнал изменения данных. По нему сервер автоматически отменяет изменения, внесенные операциями частично выполненной транзакции.

◆ *Согласованность (consistency)*. Под согласованностью понимают логическое свойство базы данных, в основе которого лежат некоторые правила, описывающие непроти-



воречивое состояние данных. В качестве такого может выступать, например, правило, гласящее, что в ходе работы транзакции, выполняющей взаимный перевод одной и той же суммы с одного счета на другой, ни с одного счета не должны исчезать деньги, а также ни на одном счете не должны появляться дополнительные фиктивные денежные суммы. Соблюдение такого рода правил требует от программиста выполнения одного условия — внутри его транзакции алгоритм перевода денег не должен выполнять несбалансированные операции обновления счетов. В задачи программиста не входит поддержание согласованности базы данных при одновременном выполнении множества транзакций. При наличии свойства согласованности система сама будет гарантировать соблюдение правил при параллельном выполнении множества транзакций. Согласованность не является самодостаточным свойством транзакции. Ее наличие подразумевает присутствие у транзакций свойств атомарности и изолированности.

♦ *Изолированность (isolation)*. Под изолированностью понимают организацию линейного порядка на множестве совместно выполняемых транзакций. Если существуют две пересекающиеся во времени транзакции  $T_1$  и  $T_2$ , то, с точки зрения пользователя, либо  $T_1$  выполнится полностью до начала  $T_2$ , либо  $T_2$  выполнится полностью до начала  $T_1$ . Изолированность позволяет избежать нарушения целостности при одновременном доступе к одним данным. Существует несколько способов практической реализации изолированности. Одним из них является использование блокировок тех данных, к которым обращаются операции в выполняемой транзакции. Заблокированы могут быть отдельное поле, запись или вся таблица БД. Попытка доступа к заблокированным данным из других транзакций приводит к переводу этих транзакций в режим ожидания до снятия блокировки. Снятие блокировки происходит в момент окончания установившей ее транзакции. В сервере БД используются особые алгоритмы по работе с множественными блокировками и защита от взаимного блокирования транзакций. Таким образом, свойство изолированности всегда выполняется без угрозы работоспособности сервера.

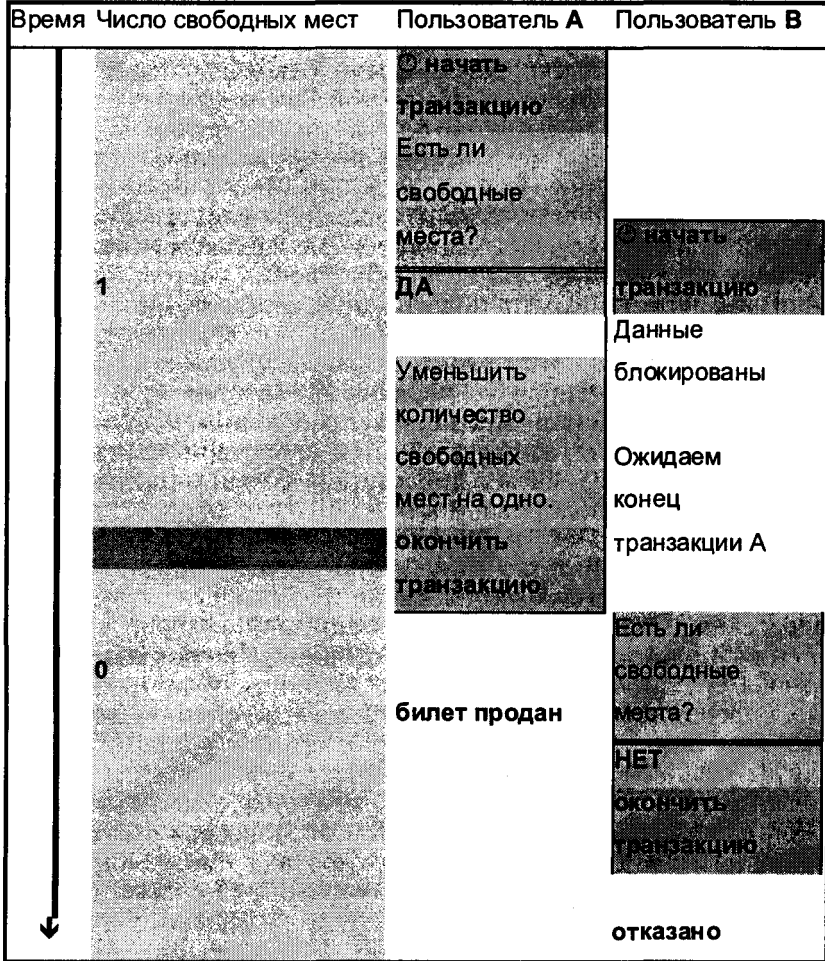


Рис. 1.1. Сохранение целостности данных при использовании транзакций

◆ *Долговечность (durability)*. Тот факт, что результаты полностью выполненной транзакции нечувствительны к краху сервера, называют долговечностью. Это свойство гарантирует, что как только транзакция успешно заверша-

ется, все сделанные в ней изменения БД немедленно переносятся из буферов сервера, расположенных в оперативной памяти, в физические файлы. Поэтому, даже если вслед за прекращением транзакции произойдет сбой в работе сервера и буфера очистятся, все результаты работы транзакции будут восстановлены. Запись результатов операций только после окончания транзакции является разумным компромиссом между требованиями сохранности данных и эффективности работы сервера. С одной стороны, сервер не тратит времени на запись в файл результатов выполнения каждой отдельной операции, а с другой — случайная очистка буферов сервера не приводит к фатальному нарушению целостности баз данных.

Для использования перечисленных свойств пользователю достаточно указывать при диалоге с сервером начало и конец транзакций. Для этого в язык манипулирования данными вводят особые операторы. Например, в большинстве языков имеется оператор начала транзакции и оператор конца транзакции. При использовании механизма транзакций алгоритм продажи авиабилетов не допускает появления ошибочных ситуаций. Клиентская программа может использовать следующий алгоритм работы:

- 1) начать транзакцию; при этом устанавливается блокировка на число свободных мест;
- 2) перед продажей билета проверить наличие мест на этот рейс; если свободных мест нет, то отказать в продаже;
- 3) уменьшить количество свободных мест на одно;
- 4) закончить транзакцию; снять блокировку;
- 5) продать билет.

В результате первому пользователю будет продан билет, а второму из-за отсутствия свободных мест будет отказано в продаже (рис. 8.5).

## ***8.2. Архитектура ЦУЭД в системе OFBIZ***

Центр управления элементами данных (ЦУЭД) в системе OFBIZ предоставляет разработчику удобное средство использования современных технологий СУБД. Задачей ЦУЭД является максимально унифицировать методы описания и манипулирования данными в составе различных

модулей системы OFBIZ. По сути, с использованием программных средств и шаблонов проектирования, составляющих основу ЦУЭД, разработчик модулей системы OFBIZ работает на уровне инфологической модели (рис. 8.1), используя для доступа к данным только понятия высокого уровня: *сущность* (Entity или элемент данных), *атрибут*, *связь*. Таким образом, он даже не обязан знать особенности и интерфейс доступа к той или иной конкретной СУБД, в которой реально хранятся данные.

Интересно отметить, что в ЦУЭД наиболее часто встречающиеся последовательности действий по обработке данных воплощены в виде шаблонов проектирования, которые были получены на основании глубокого анализа алгоритмов и структур данных в различных информационных системах и тщательно рассмотрены в нескольких книгах<sup>39</sup>. Поэтому можно считать, что в ЦУЭД воплощено профессиональное знание многих специалистов в области эффективного хранения и обработки больших объемов корпоративных данных.

Среди всех использованных шаблонов два являются основополагающими составляющими архитектуры ЦУЭД.

◆ *Generic value*. Экземпляры различных типов элементов данных представляются для разработчика как экземпляры одного и того же Java-класса `GenericValue`. Получение и модификация значений атрибутов элемента данных производится с помощью общих методов (например, `getString`), имеющих один параметр — имя атрибута. Таким образом, класс `GenericValue` представляет собой ассоциативный массив особого рода.

◆ *XML Entity definition file*. Определения структуры элементов данных и взаимосвязей между ними (т.е. модель данных) загружаются на старте OFBIZ из текстовых XML-файлов. Они содержат имена всех типов элементов данных, имена и типы атрибутов, с указанием соответствующих таблиц и полей реляционных СУБД, а также все характеристики связи между разными элементами данных — тип связанного элемента данных, тип связи (один к одному,

---

<sup>39</sup> Например, Alur, Crupi, Malks «CoreJ2EE Patterns». Информацию по этим шаблонам также можно получить на сайте <http://developer.java.sun.com/developer/restricted/patterns/J2EEPatternsAtAGlance.html>.

один ко многим), внешние ключи и уникальное имя связи<sup>40</sup>. Кроме того, в файлах содержится вся необходимая информация по правилам автоматического преобразования стандартных типов данных языка Java в типы данных используемой реляционной СУБД. В ЦУЭД реализованы алгоритмы, обеспечивающие доступ к СУБД на основании информации из XML-файлов, а также процедуры проверки корректности выполняемых операций с данными. Все это позволяет освободить разработчика модуля от необходимости создания собственного кода для сохранения/восстановления значений Java-объектов из БД, соответствующих элементам данных.

На рис. 8.6 представлена взаимосвязь различных служебных классов и интерфейсов OFBIZ, составляющих в совокупности ЦУЭД.

Все операции по созданию, поиску и модификации элементов данных выполняются клиентами с помощью методов класса *общий делегат* (Generic delegator)<sup>41</sup>. Методы, предоставляемые общим делегатом, достаточно абстрактны и не связаны с типом конкретной СУБД. Это позволяет легко создавать и модифицировать специфический программный код для хранения элементов данных в различных СУБД, совершенно не затрагивая прикладные алгоритмы в службах OFBIZ. На основании конкретной конфигурации из файла:

```
%OFBIZ_HOME\framework\entity\config\entityengine.xml
```

общий делегат перенаправляет запросы в конкретный класс-исполнитель (Helper). Каждый такой класс-исполнитель умеет общаться с определенным типом СУБД или другим типом внешнего долговременного хранилища данных для выполнения затребованной операции с элементами данных. Для всех подобных внешних систем долговременного хранения данных в системе OFBIZ используется общее название — *источник данных* (DataSource). Поэтому правильно будет сказать, что класс-исполнитель связан с определенным источником данных.

---

<sup>40</sup> Это позволяет вводить несколько различных видов связей между одними и теми же типами элементов данных.

<sup>41</sup> В системе может быть определено несколько общих делегатов, имеющих уникальное символическое имя.

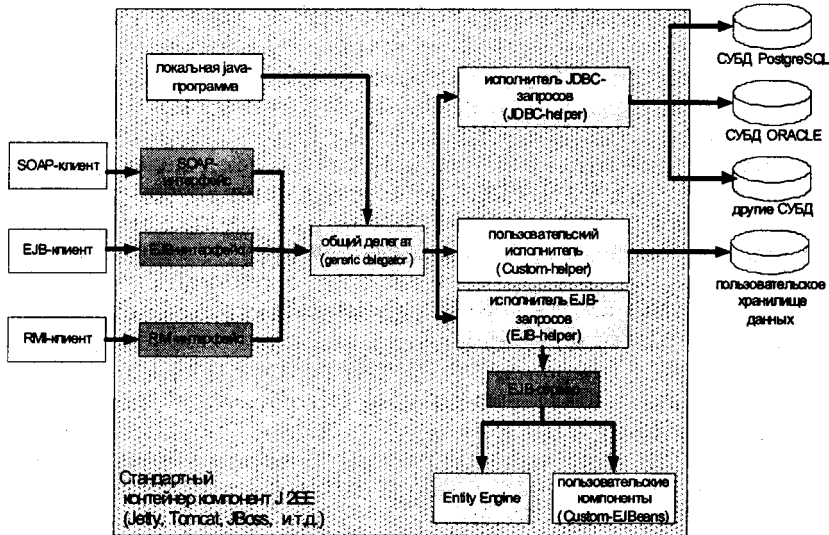


Рис. 8.6. Общая архитектура центра управления элементами данных

Нужно добавить, что в системе OFBIZ принято, чтобы каждый тип элемента данных логически принадлежал ровно к одной *группе* (Group). На этапе конфигурации системы в определении каждого экземпляра общего делегата устанавливается своя связь между группой элементов данных и источником данных. Все элементы данных, принадлежащие к одной и той же группе, хранятся в одном источнике данных. Таким образом, в зависимости от используемого общего делегата экземпляры элемента данных одного и того же типа могут реально находиться в различных источниках данных. Прикладной программист может просто переключиться с реляционной СУБД на другие технологии долговременного хранения данных (например, на использование EJB-сервера) путем изменения конфигурации в XML-файле %OFBIZ\_HOME%\framework\entity\config\entityengine.xml, или переходя на использование общего делегата с другим именем.

### **8.3. Правила определения элементов данных и связей между ними (модель данных модуля OFBIZ)**

Модель данных, необходимых для работы модуля в системе OFBIZ, содержится в четырех XML-файлах, располагающихся обычно в подкаталоге каждого компонента entitydef:

1) файл, содержащий СУБД-независимое определение элементов данных, их атрибутов и связей (обычно он называется entitymodel.xml);

2) файл для распределения по группам вновь определенных типов элементов данных (entitygroup.xml);

3) файл с определениями специфичных правил реакции системы на изменение атрибутов того или иного элемента данных (eecas.xml, Entity Event-Condition-Action);

4) файл, определяющий отображение общих типов данных, используемых в ЦУЭД, на типы данных, специфичные для конкретной СУБД.

В большинстве случаев разработчику модуля достаточно создать файлы entitymodel.xml и entitygroup.xml, а также внести небольшие изменения в файл конфигурации модуля ofbiz-component.xml.

#### **8.3.1. Файл с определениями элементов данных (entitymodel.xml)**

XML-файл с определениями элементов данных строится в соответствии с XML-схемой

`%OFBIZ_HOME%\framework\entity\dtd\entitymodel.xsd`.

Структура этой схемы в виде UML-диаграммы представлена на рис. 8.7.

XML-тег <entity> может содержать внутри себя различные подтеги, назначение и состав атрибутов которых раскрывается в следующих таблицах.

В свою очередь подтег <field> может содержать подчиненный подтег <validate> с атрибутом name. В нем определяется имя java-метода для проверки корректности значения поля. Отметим, что такой метод автоматически не вызывается во время обычных операций с экземплярами элемента данных. Вызов метода происходит лишь в административных приложениях типа «Entity Data Maintenance».

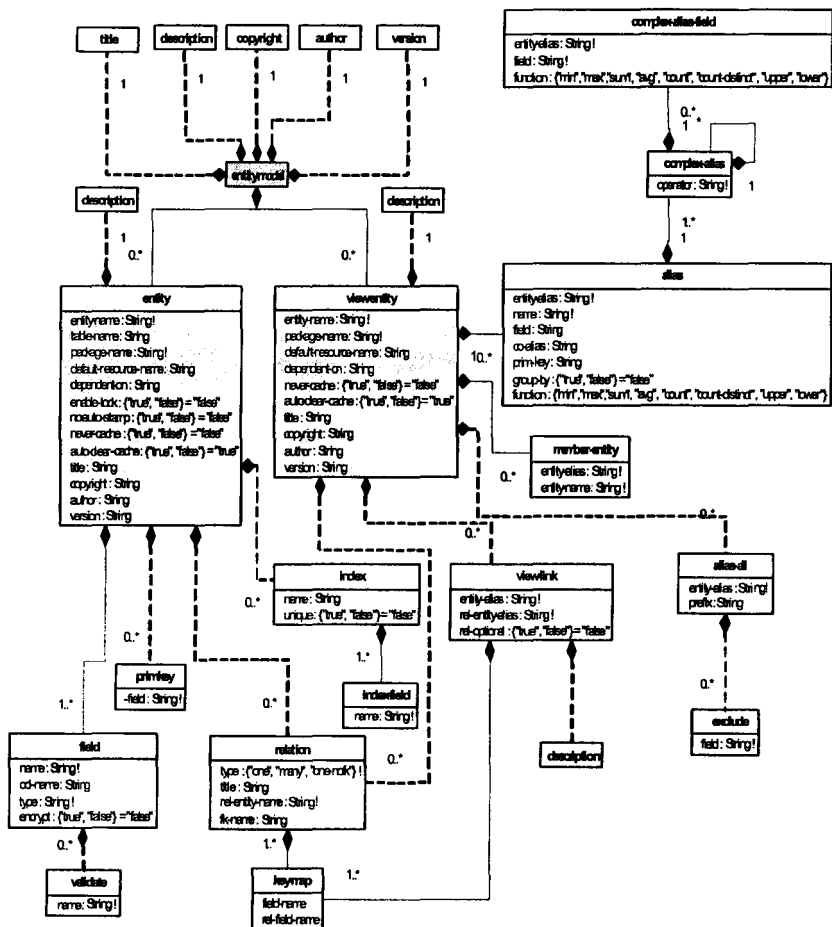


Рис. 8.7. Взаимосвязь элементов в файле определения элементов данных. Структура элемента данных определяется в атрибутах XML-тега <entity>

Подтег <relation> может содержать один или несколько подтегов <key-map>. Каждый такой подтег используется для указания соответствия между полем в определяемом элементе данных и полем в связанном элементе данных. <key-map> имеет два атрибута: **field-name** и **rel-field-name**, в



## Описание атрибутов тега <entity>

Имя атрибута	Требуется обязательно?	Назначение
1	2	3
entity-name	Да	Имя элемента данных, по которому к нему ссылаются при использовании Entity Engine Java API и других мест в системе OFBIZ
table-name	Нет	Имя таблицы БД, которая соответствует определяемому элементу данных. Этот атрибут является необязательным и если не определен, то имя таблицы строится на основе имени элемента данных
package-name	Да	Имя пакета (package) в котором располагается определяемый элемент данных. Использование пакетов позволяет структурировать большое число элементов данных
dependent-on	Нет	Выявляет какой-то элемент данных, от которого зависит определяемый элемент. Этот атрибут может быть использован для определения иерархической структуры элементов данных
enable-lock	Нет	Определяет, требуется ли механизм оптимистичной блокировки при работе с определяемым элементом данных. В случае использования механизма оптимистичной блокировки в составе элемента данных должно присутствовать поле <code>lastUpdatedStamp</code> . Оно будет использоваться для сохранения времени последней модификации экземпляров этого элемента данных. Если такого поля нет, то будет выдана исключительная ситуация <code>EntityLockedException</code>
never-cache	Нет	Если атрибут установлен в значение «true», то кеширование экземпляров этого элемента данных будет запрещено. Также не будет проводиться автоматической очистки кеша, а любая попытка использования методов работы с кешем для этого элемента данных будет приводить к возникновению исключительной ситуации
title	Нет	Заголовок для элемента данных. Если значение не установлено, то используется значение по умолчанию — значение для модели, в которой располагается этот элемент данных.

1	2	3
copyright	Нет	Текст соглашения об авторском праве для элемента данных. Если значение не установлено, то используется значение по умолчанию — значение для модели, в которой располагается этот элемент данных
author	Нет	Автор элемента данных. Если значение не установлено, то используется значение по умолчанию — значение для модели, в которой располагается этот элемент данных
version	Нет	Версия элемента данных. Если значение не установлено, то используется значение по умолчанию — значение для модели, в которой располагается этот элемент данных

Таблица 8.2

### Имя подтега

Имя подтега	Количество	Назначение
description	0 или 1	Подробное описание элемента данных. Разрешается использовать произвольную текстовую строку. Если значение не установлено, то используется значение по умолчанию — значение для модели, в которой располагается этот элемент данных
field	1 или много	Используется для определения полей в составе элемента данных
prim-key	0 или много	Используется для указания ключевых полей. Разрешается использовать несколько идущих друг за другом подтегов <prim-key>, определяющих различные кандидатные ключи
relation	0 или много	Используется для определения взаимосвязей между элементами данных

которых задаются имена соответствующих полей определяемого и связанного элемента данных.

Особого внимания заслуживает XML-тег <view-entity>. С помощью этого средства разработчик может определить так называемые «виртуальные» элементы данных. Это понятие в системе OFBIZ тождественно понятию «представ-

## Описание атрибутов подтега &lt;field&gt;

Имя атрибута	Требуется обязательно?	Назначение
name	Да	Имя поля, по которому к нему ссылаются в коде на языке Java и в других местах системы OFBIZ
col-name	Нет	Имя соответствующего поля таблицы БД. Если значение не указано явно, то выводится на основании имени поля элемента данных
type	Да	Тип данных определяемого поля. Значение должно быть одним из типов, перечисленных для данного источника данных в XML-файле (fieldtypemodel). Значение используется во время работы для определения нужного типа данных в языке Java или SQL

## Описание атрибутов подэлемента prim-key

Имя атрибута	Требуется обязательно?	Назначение
field	Да	Часть имени поля, которая войдет в primary key

## Описание атрибутов подтега &lt;relation&gt;

Имя атрибута	Требуется обязательно?	Назначение
1	2	3
type	Да	Определяет тип связи между элементами данных. Тип устанавливает кардинальность связи на стороне связанного элемента данных и необходимость создания внешнего ключа для связи с кардинальностью «1:1». Разрешенные типы связи «one», «one-tofk», или «many»
title	Нет	Этот атрибут позволяет задать дополнительный префикс перед названием элемента данных при формировании имени связи. Такая возможность полезна в том случае, когда между одними и теми же элементами данных установлено несколько различных видов связи

1	2	3
rel-entity-name	Да	Имя связанного элемента данных. Навигация по связи в системе OFBIZ осуществляется в направлении от текущего элемента данных к связанному элементу данных
fk-name	Нет	Позволяет явно задать имя поля, являющегося внешним ключом. Хотя имя внешнего ключа может быть сгенерировано автоматически на основании имени связи, на практике этой возможностью пользоваться не рекомендуется, так как во многих СУБД существует ограничение на длину имени ключа и индекса (например, всего 18 символов), а также требование на уникальность имени внешнего ключа в пределах всей базы данных

Таблица 8.6

## Имя подэлемента

Имя подэлемента	Количество	Назначение
1	2	3
description	0 или 1	Подробное описание элемента данных. Разрешается использовать произвольную текстовую строку. Если значение не установлено, то используется значение по умолчанию — значение для модели, в которой располагается этот элемент данных
member-entity	1 или много	Используется для определения того «реального» элемента данных, который является частью определяемого «виртуального» элемента данных. «Реальные» элементы данных, являющиеся частью «виртуального», носят название элемент-член (member entity) и получают псевдоним, по которому на них можно ссылаться в определении «виртуального» элемента данных. Такая практика позволяет включать в состав одного «виртуального» элемента данных один и тот же «реальный» элемент данных несколько раз

1	2	3
alias	1 к многим	Атрибут используется для переименования полей «реального» элемента данных, когда он входит составной частью в «виртуальный» элемент данных. Если переименования поля элемента-члена не производится, то оно включается в состав «виртуального» элемента данных с прежним именем
view-link	1 или много	Определяет правила связи различных элементов-членов в составе определяемого «виртуального» элемента данных. Определение связи использует псевдонимы элементов-членов и установленные с помощью тега <key-maps> соответствия между их полями, подобно тому, как определяется связь между элементами данных в теге <relation>
relation	0 к многим	Используется для объявления отношений

Таблица 8.7

### Атрибуты подэлемента member-entity

Имя атрибута	Требуется обязательно?	Назначение
entity-alias	Да	Используется для установления ссылок и связей
entity-name	Да	Имя

Таблица 8.8

### Атрибуты подэлемента alias

Имя атрибута	Требуется обязательно?	Назначение
entity-alias	Да	Используется для установления ссылок и связей
name	Да	Имя
field	Нет	Имя поля
prim-key	Нет	Установлен ие ключа primary key
group-by	Нет	Используется в операциях группировки
function	Нет	Используется для ссылки на функцию для проведения групповых операций

## Атрибуты подэлемента view-link

Имя атрибута	Требуется обязательно?	Назначение
entity-alias	Да	Ссылка на источник.
rel-entity-alias	Да	Ссылка на адресат
rel-optional	Нет	Указатель на возможность связи

ление» (view) в реляционных базах данных: «виртуальный» элемент данных является комбинацией атрибутов других элементов данных. Комбинация осуществляется с помощью реляционной операции объединения JOIN. Использование «виртуального» элемента данных облегчает работу с взаимосвязанными экземплярами различных элементов данных, либо скрывает от разработчика часть атрибутов «реальных» элементов данных. Для хранения значений атрибутов «виртуального» элемента данных отдельной таблицы обычно не создается, а информация извлекается из таблиц, соответствующих «реальным» элементам данных согласно определению в XML-теге <view-entity>. Атрибуты XML-тега <view-entity> почти такие же, как и у тега <entity>. Тег <view-entity> имеет имя, назначенный пакет, описание, соглашение об авторских правах, автора, версию и т.д. Однако в этом теге отсутствует указание таблицы, в которой хранятся данные. «Виртуальный» элемент данных должен быть также назначен в определенную группу и это назначение проводится как обычно в XML-файле entity-group.xml. Отметим, что все те «реальные» элементы данных, из которых состоит «виртуальный» элемент данных, должны находиться в одной базе данных, хотя и необязательно в одной группе элементов.

Большие отличия заметны в составе подтегов XML-тега <view-entity>. Он может содержать такие подэлементы (см. табл. 8.6).

#### 8.4. Примеры определения модели

В качестве примера определения элементов данных рассмотрим содержимое файла:

```
%OFBIZ_HOME%\framework\example\entitydef\entitymodel.xml.
```

```

<?xml version="1.0" encoding="UTF-8"?>
<entitymodel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/entitymodel.xsd">
  <!-- ===== -->
  <!-- ===== Defaults ===== -->
  <!-- ===== -->
  <title>Entity of an Open For Business Project Component</title>
  <description>None</description>
  <copyright>Copyright (c) 2001-2004 The Open For Business Project -
www.ofbiz.org</copyright>
  <author>None</author>
  <version>1.0</version>
  <!-- ===== -->
  <!-- ===== Data Model ===== -->
  <!-- The modules in this file are as follows: -->
  <!-- - org.ofbiz.example -->
  <!-- ===== -->
  <!-- ===== -->
  <!-- org.ofbiz.example -->
  <!-- ===== -->
  <entity entity-name="Example" package-name="org.ofbiz.example" title="Example
Entity">
  <field name="exampleId" type="id-ne" col-name="EXAMPLE_KEY_FIELD_ONE "/>

  <field name="exampleTypeId" type="id-ne"/>
  <field name="partyId" type="id"/>
  <field name="exampleName" type="name"/>
  <field name="description" type="description"/>
  <field name="comments" type="comment"/>
  <field name="exampleSize" type="numeric"/>
  <field name="exampleDate" type="date-time"/>
  <prim-key field="exampleId"/>
  <relation type="one" fk-name="EXMPL_TYP" rel-entity-name="ExampleType">
    <key-map field-name="exampleTypeId"/>
  </relation>
  <relation type="one" fk-name="EXMPL_PTY" rel-entity-name="Party">
    <key-map field-name="partyId"/>
  </relation>
</entity>
  <entity entity-name="ExampleType" package-name="org.ofbiz.example" title="Example
Type Entity">
  <field name="exampleTypeId" type="id-ne"/>
  <field name="description" type="description"/>
  <prim-key field="exampleTypeId"/>
</entity>
</entitymodel>

```

**Рис. 8.8.** Пример определения структуры элементов данных в файле entitymodel.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<entitygroup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/entitygroup.xsd">
  <!-- ===== -->
  <!-- org.ofbiz.example -->
  <!-- ===== -->
  <entity-group group="org.ofbiz" entity="Example"/>
  <entity-group group="org.ofbiz" entity="ExampleType"/>
</entitygroup>

```

**Рис. 8.9.** Включение в группу нового элемента данных в файле entitygroup.xml

В этом файле определено два элемента данных с именами Example и Example-Type, соответственно.

Из приведенного на рис. 8.8 примера видно, что имя поля реляционной таблицы явно указано только в определении атрибута exampleId. В подавляющем большинстве случаев имена полей таблицы автоматически производятся системой OFBIZ на основании имен соответствующих атрибутов и принятых в системе соглашений по именованию атрибутов, таблиц и их полей. По этим соглашениям в XML-файле имена атрибутов элемента данных должны быть написаны в соответствии с правилами языка Java на именование полей класса: все буквы строчные, кроме

```

<?xml version="1.0" encoding="UTF-8"?>
<fieldtypemodel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/fieldtypemodel.xsd">
  <!-- ===== field-type-def ===== -->
  <!-- General Types -->
  <field-type-def type="blob" sql-type="BLOB" java-type="java.lang.Object"/>
  <field-type-def type="date-time" sql-type="TIMESTAMP" java-
type="java.sql.Timestamp"/>
  <field-type-def type="date" sql-type="DATE" java-type="java.sql.Date"/>
  <field-type-def type="time" sql-type="TIME" java-type="java.sql.Time"/>
  <field-type-def type="currency-amount" sql-type="DOUBLE" java-type="Double">
    <validate method="isSignedDouble"/>
  </field-type-def>
  <field-type-def type="floating-point" sql-type="DOUBLE" java-type="Double">
    <validate method="isSignedDouble"/>
  </field-type-def>
  <field-type-def type="numeric" sql-type="NUMERIC(18,0)" java-type="Long">
    <validate method="isSignedLong"/>

```

**Рис. 8.10.** Правила отображения типов данных и проверок на корректность для СУБД Derby в файле Fieldtypederby.xml



```

</field-type-def>
<field-type-def type="id" sql-type="VARCHAR(20)" java-type="String"/>
<field-type-def type="id-long" sql-type="VARCHAR(60)" java-type="String"/>
<field-type-def type="id-vlong" sql-type="VARCHAR(255)" java-type="String"/>
<field-type-def type="indicator" sql-type="CHAR(1)" java-type="String"/>
<field-type-def type="very-short" sql-type="VARCHAR(10)" java-type="String"/>
<field-type-def type="short-varchar" sql-type="VARCHAR(60)" java-type="String"/>
<field-type-def type="long-varchar" sql-type="VARCHAR(255)" java-type="String"/>
<field-type-def type="very-long" sql-type="CLOB" java-type="String"/>
<field-type-def type="comment" sql-type="VARCHAR(255)" java-type="String"/>
<field-type-def type="description" sql-type="VARCHAR(255)" java-type="String"/>
<field-type-def type="name" sql-type="VARCHAR(100)" java-type="String"/>
<field-type-def type="value" sql-type="VARCHAR(255)" java-type="String"/>
<!-- Specialized Types -->
<field-type-def type="credit-card-number" sql-type="VARCHAR(255)" java-type="String">
  <validate method="isAnyCard"/>
</field-type-def>
<field-type-def type="credit-card-date" sql-type="VARCHAR(20)" java-type="String">
  <validate method="isDateAfterToday"/>
</field-type-def>
<field-type-def type="email" sql-type="VARCHAR(255)" java-type="String">
  <validate method="isEmail"/>
</field-type-def>
<field-type-def type="url" sql-type="VARCHAR(255)" java-type="String"/>
<field-type-def type="id-ne" sql-type="VARCHAR(20)" java-type="String">
  <validate method="isNotEmpty"/>
</field-type-def>
<field-type-def type="id-long-ne" sql-type="VARCHAR(60)" java-type="String">
  <validate method="isNotEmpty"/>
</field-type-def>
<field-type-def type="id-vlong-ne" sql-type="VARCHAR(255)" java-type="String">
  <validate method="isNotEmpty"/>
</field-type-def>
<field-type-def type="tel-number" sql-type="VARCHAR(60)" java-type="String">
  <validate method="isInternationalPhoneNumber"/>
</field-type-def>
</fieldtypemodel>

```

**Рис. 8.10.** Правила отображения типов данных и проверок на корректность для СУБД Derby в файле Fieldtypederby.xml (окончание)

первых букв слов, составляющих имя. Пробелы в имени атрибутов не допускаются. В имени элементов данных также все буквы строчные, кроме первых букв слов, составляющих имя. Но в имени элемента данных первая буква обязательно заглавная и разрешается использовать пробелы между словами.

По правилам системы OFBIZ автоматически сгенерированные или указанные разработчиком имена таблиц и их полей могут содержать только заглавные буквы, а отдельные слова в имени разделяются символом подчеркивания «\_».

Следовательно, например, SampleEntity и fieldOne соответствуют именам таблицы SAMPLE\_ENTITY и поля FIELD\_ONE.

Само по себе определение элемента данных в файле entitymodel.xml еще не делает его включенным в общую модель данных системы OFBIZ. Разработчику обязательно требуется распределить все новые типы элементов данных по группам в отдельном XML-файле, который чаще всего носит название entity-group.xml. Структура этого файла достаточно проста и может быть проанализирована самостоятельно из примера на рис. 8.9 или по соответствующей XML-схеме:

```
%OFBIZ_HOME%\framework\entity\dtd\entitygroup.xsd.
```

Наряду с определением правил отображения из типов языка java в типы СУБД в этом XML-файле можно указать специальные методы (validator), которые будут автоматически вызываться для проверки правильности значений. Все такие методы реализованы в составе класса org.ofbiz.base.UtilValidate и имеют похожую структуру (например, Boolean isValid(String in)). Общие для всех модулей системы OFBIZ типы данных и правила их преобразования к типам данных различных СУБД находятся в каталоге

```
%OFBIZ_HOME%\ framework\entity\ fieldtype.
```

В качестве примера на рис. 8.10 показано, как задается преобразование в типы данных СУБД Derby (эта СУБД используется по умолчанию в системе OFBIZ).

## ***8.5. Инициализация описаний элементов данных на старте модуля***

В составе конфигурации сложного компонента системы OFBIZ может быть создано несколько файлов с определениями элементов данных. Чтобы на этапе инициализации компонента об этих файлах стало известно, системе требу-

```

<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="common" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/ofbiz-
component.xsd">
  <resource-loader name="main" type="component"/>
  <classpath type="jar" location="build/lib/*"/>
  <classpath type="dir" location="config"/>
  <classpath type="dir" location="script"/>
  <entity-resource type="model" reader-name="main" loader="main"
location="entitydef/entitymodel.xml"/>
  <entity-resource type="group" reader-name="main" loader="main"
location="entitydef/entitygroup.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/CommonTypeData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/CountryCodeData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/CurrencyData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/GeoData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/GeoData_BR.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/GeoData_US.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/LanguageData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/UnitData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/PeriodData.xml"/>
  <service-resource type="model" loader="main" location="servicedef/services.xml"/>
  <service-resource type="model" loader="main" location="servicedef/services_test.xml"/>
  <service-resource type="group" loader="main" location="servicedef/groups_test.xml"/>
  <service-resource type="eca" loader="main" location="servicedef/secas_test.xml"/>
  <service-resource type="mca" loader="main" location="servicedef/smcas_test.xml"/>
</ofbiz-component>

```

**Рис. 8.11.** Указание файлов определения элементов данных в конфигурационном файле компонента ofbiz-component.xml

ется перечислить их в главном конфигурационном файле компонента ofbiz-component.xml. На рис. 8.11 представлен главный конфигурационный файл компонента common:

**%OFBIZ\_HOME%\framework\common\ofbiz-component.xml,**

в котором жирным цветом выделены строки с перечислением используемых файлов определения элементов данных и групп.

Кроме того, в файле конфигурации компонента можно указать, каким образом выполняется начальная загрузка данных в ЦУЭД. Для этого используется XML-элемент `entity-resource` у которого значение атрибута `type` равно `data`. В атрибуте `location` у такого XML-элемента указывается имя XML-файла, содержащего исходные данные.

## 8.6. Способы манипулирования элементами данных в Java-классах

С точки зрения программиста, все операции с элементами данных и взаимодействие с ЦУЭД осуществляются в системе OFBIZ с использованием трех служебных Java-классов из пакета `org.ofbiz.entity`:

- ◆ `GenericDelegator`;
- ◆ `GenericValue`;
- ◆ `GenericPK`.

Класс `GenericDelegator` реализует интерфейс `DelegatorInterface`, который является не чем иным, как *общим делегатом* из рис. 8.6 и предоставляет операции `create`, `find`, `store` (а также ряд других) для манипулирования различными типами элементов данных. Сводка основных методов интерфейса `Delegator-Interface`, реализованных в классе `GenericDelegator`, представлена в табл. 8.10.

Таблица 8.10

### Основные методы

Очистка кеша ранее загруженных элементов данных
<code>clearAllCacheLinesByDummyPK(dummyPKs : java.util.Collection)</code>
<code>clearAllCacheLinesByValue( values : java.util.Collection)</code>
<code>clearAllCaches()</code>
<code>clearAllCaches( distribute : boolean)</code>
<code>clearCacheLine( primaryKey : GenericPK)</code>
<code>clearCacheLine( primaryKey : GenericPK, distribute : boolean)</code>
<code>clearCacheLine( value : genericValue)</code>
<code>clearCacheLine( entityName : String, fields : java.util.Map)</code>
<code>clearCacheLineByCondition( entityName : String, condition : EntityCondition)</code>
<code>clearCacheLineFlexible( dummyPK : GenericEntity)</code>
<code>clearCacheLineFlexible( dummyPK : GenericEntity, distribute : boolean)</code>

**Создание новых экземпляров элементов данных****create( primaryKey : GenericPK ) : genericValue****create( primaryKey : GenericPK, doCacheClear : boolean ) : genericValue****create( value : genericValue ) : genericValue****create( value : genericValue, doCacheClear : boolean ) : genericValue****create( entityName : String, fields : java.util.Map ) : genericValue****createOrStore( value : genericValue ) : genericValue****createOrStore( value : genericValue, doCacheClear : boolean ) : genericValue****Поиск экземпляров элементов данных****findAll( entityName : String ) : java.util.List****findAll( entityName : String, orderBy : java.util.List ) : java.util.List****findAllByPrimaryKeys( primaryKeys : java.util.Collection ) : java.util.List****findAllByPrimaryKeysCache( primaryKeys : java.util.Collection ) : java.util.List****findAllCache( entityName : String ) : java.util.List****findAllCache( entityName : String, orderBy : java.util.List ) : java.util.List****findByAnd( modelEntity : ModelEntity, fields : java.util.Map, orderBy : java.util.List ) : java.util.List****findByAnd( entityName : String, expressions : java.util.List ) : java.util.List****findByAnd( entityName : String, expressions : java.util.List, orderBy : java.util.List ) : java.util.List****findByAnd( entityName : String, fields : java.util.Map ) : java.util.List****findByAnd( entityName : String, fields : java.util.Map, orderBy : java.util.List ) : java.util.List****findByAndCache( entityName : String, fields : java.util.Map ) : java.util.List****findByAndCache( entityName : String, fields : java.util.Map, orderBy : java.util.List ) : java.util.List****findByCondition( entityName : String, entityCondition : EntityCondition, fieldsToSelect : java.util.Collection, orderBy : java.util.List ) : java.util.List****findByLike( entityName : String, fields : java.util.Map ) : java.util.List****findByLike( entityName : String, fields : java.util.Map, orderBy : java.util.List ) : java.util.List****findByOr( findByOr : String, expressions : java.util.List ) : java.util.List****findByOr( findByOr : String, expressions : java.util.List, orderBy : java.util.List ) : java.util.List****findByPrimaryKey( primaryKey : GenericPK ) : genericValue****findByPrimaryKey( entityName : String, fields : java.util.Map ) : genericValue****findByPrimaryKeyCache( primaryKey : GenericPK ) : genericValue**

**findByPrimaryKeyCache**( entityName : String, fields : java.util.Map) : genericValue

**findByPrimaryKeyPartial**( primaryKey : GenericPK, keys : java.util.Set) : genericValue

**findCountByAnd**() : genericValue

**findCountByCondition**() : genericValue

**findListIteratorByCondition**( entityName : String, whereEntityCondition : EntityCondition, havingEntityCondition : EntityCondition, fieldsToSelect : java.util.Collection, orderBy : java.util.List, findOptions) : EntityListIterator

### Получение служебной информации

**getCache**()

**getDelegatorName**() : String

**getEntityType**( entity : ModelEntity, type : String)

**getEntityTypeNames**( entity : ModelEntity) : java.util.Collection

**getEntityGroupName**( entityName : String) : String

**getEntityHelper**( entity : ModelEntity)

**getEntityHelper**( entityName : String)

**getEntityHelperName**( entity : ModelEntity) : String

**getEntityHelperName**( entityName : ModelEntity) : String

**getFromPrimaryKeyCache**( primaryKey : GenericPK) : genericValue

**getGroupHelperName**( groupName : String) : String

**getModelGroupReader**()

**getModelReader**()

### Поиск связанных по какому-то отношению экземпляров элементов данных

**getMultiRelation**( value : genericValue, relationNameOne : String, relationNameTwo : String) : java.util.List

**getMultiRelation**( value : genericValue, relationNameOne : String, relationNameTwo : String, orderBy : java.util.List) : java.util.List

**getNextSeqId**( seqName : String) : String

**getNextSeqId**( seqName : String, staggerMax : long) : String

**getNextSeqIdLong**( seqName : String) : long

**getNextSeqIdLong**( seqName : String, staggerMax : long) : long

**getRelated**( relationName : String, value : genericValue) : java.util.List

**getRelated**( relationName : String, byAndFields : java.util.Map, value : genericValue) : java.util.List

**getRelatedByAnd**( relationName : String, byAndFields : java.util.Map, value : genericValue) : java.util.List

<b>getRelatedCache</b> ( relationName : String, value : genericValue) : java.util.List
<b>getRelatedDummyPK</b> ( relationName : String, byAndFields : java.util.Map, value : genericValue) : GenericPK
<b>getRelatedOne</b> ( relationName : String, value : genericValue) : genericValue
<b>getRelatedOneCache</b> ( relationName : String, value : genericValue) : genericValue
<b>getRelatedOrderBy</b> ( relationName : String, orderBy : java.util.List, value : genericValue) : genericValue
<b>Создание новых служебных классов</b>
<b>makePK</b> ( entityName : String, fields : java.util.Map) : GenericPK
<b>makeValidValue</b> ( entityName : String, fields : java.util.Map) : genericValue
<b>makeValue</b> ( entityName : String, fields : java.util.Map) : genericValue
<b>Обновление данных</b>
<b>refresh</b> ( value : genericValue)
<b>refresh</b> ( value : genericValue, doCacheClear : boolean)
<b>refreshSequencer</b> ()
<b>Удаление экземпляров элементов данных</b>
<b>removeAll</b> ( dummyPKs : java.util.List) : int
<b>removeAll</b> ( dummyPKs : java.util.List, doCacheClear : boolean) : int
<b>removeByAnd</b> ( entityName : String, fields : java.util.Map) : int
<b>removeByAnd</b> ( entityName : String, fields : java.util.Map, doCacheClear : boolean) : int
<b>removeByCondition</b> ( entityName : String, condition : EntityCondition) : int
<b>removeByCondition</b> ( entityName : String, condition : EntityCondition, doCacheClear : boolean) : int
<b>removeByPrimaryKey</b> ( primaryKey : GenericPK) : int
<b>removeByPrimaryKey</b> ( primaryKey : GenericPK, doCacheClear : boolean) : int
<b>removeRelated</b> ( relationName : String, value : genericValue) : int
<b>removeRelated</b> ( relationName : String, value : genericValue, doCacheClear : boolean) : int
<b>removeValue</b> ( value : genericValue) : int
<b>removeValue</b> ( value : genericValue, doCacheClear : boolean) : int
<b>setNextSubSeqId</b> ()
<b>setSequencer</b> ( sequencer : SequenceUtil)
<b>Долговременное сохранение элементов данных</b>
<b>store</b> ( value : genericValue) : int
<b>store</b> ( value : genericValue, doCacheClear : boolean) : int
<b>storeAll</b> ( values : java.util.List) : int
<b>storeAll</b> ( values : java.util.List, doClearCache : boolean) : int

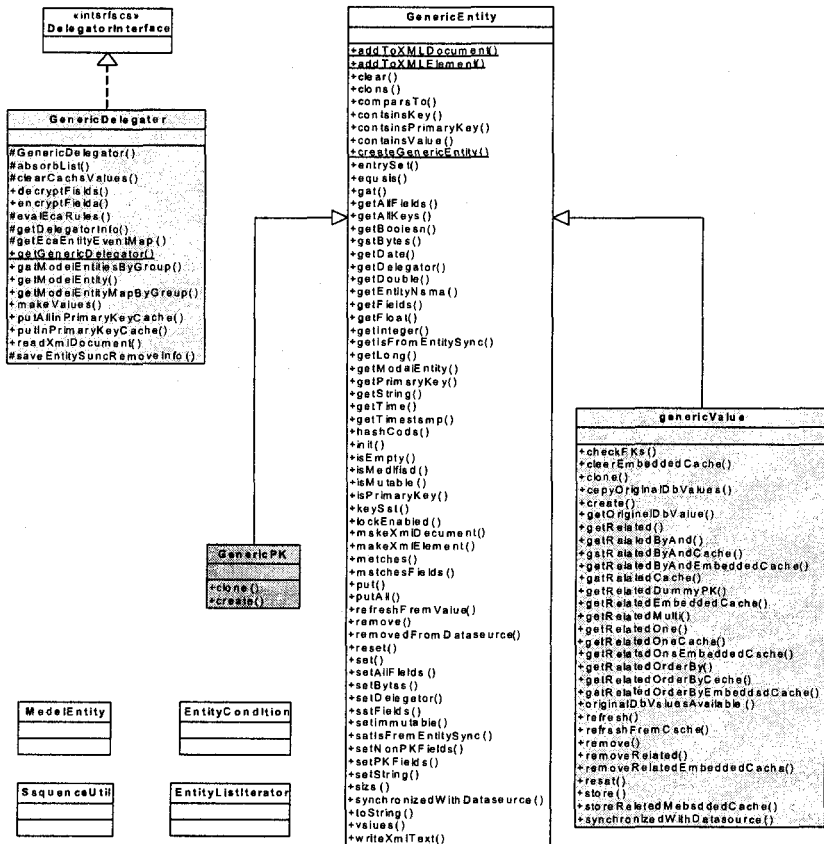


Рис. 8.12. UML-диаграмма классов и интерфейсов для работы с ЦУЭД

Как уже упоминалось, экземпляр класса **GenericValue** одинаковым образом представляет для программиста экземпляр *любого* типа элемента данных. Класс **GenericValue** является наследником класса **GenericEntity**, а тот, в свою очередь, реализует стандартный Java-интерфейс для работы с ассоциативными массивами **Map**. Поэтому доступ к значению любого атрибута экземпляра класса **GenericValue**



осуществляется по имени атрибута. Интересно отметить, что после своего создания экземпляр `GenericValue` содержит ссылку на общий делегат, который его создал. Поэтому через эту ссылку можно выполнять операции сохранения, удаления или модификации элементов данных без прямого обращения к классу `GenericDelegator`. Таким образом, в классе `GenericValue` доступны почти такие же методы, что и в интерфейсе `DelegatorInterface`, за одним исключением — в методах класса `GenericValue` отсутствует параметр с именем `value`.

Наконец, класс `GenericPK` выражает понятие «первичный ключ» и является ассоциативным массивом, в котором хранятся имена и значения ключевых полей определенного типа элементов данных. Класс `GenericPK` является наследником класса `GenericEntity` (рис. 8.12).

## 8.7. Примеры кода

Рассмотрим на ряде примеров исходного кода различных компонентов системы OFBIZ основные приемы использования ЦУЭД. Фрагменты кода, напрямую относящиеся к работе с элементами данных, выделены жирным шрифтом.

1. Пример создания нового экземпляра элемента данных типа `NoteData` в службе `createNote` компонента `common`. В методе `makeValue` значения атрибутов экземпляра инициализируются из заранее подготовленного ассоциативного массива с именем `fields`.

```
public static Map createNote(DispatchContext ctx, Map context)
{
    GenericDelegator delegator = ctx.getDelegator();
    GenericValue userLogin = (GenericValue) context.get("userLogin");

    Timestamp now = UtilDateTime.nowTimestamp();

    String partyId = (String) context.get("partyId");
    String noteName = (String) context.get("noteName");
    String note = (String) context.get("note");
    String noteId = delegator.getNextSeqId("NoteData");

    // check for a party id
    if (partyId == null)
    {
        if (userLogin != null && userLogin.get("partyId") != null)
            partyId = userLogin.getString("partyId");
    }
}
```

```

Map fields = UtilMisc.toMap("noteId", noteId, "noteName", noteName,
"noteInfo", note, "noteParty", partyId, "noteDateTime", now);
try
{
    GenericValue newValue = delegator.makeValue("NoteData", fields);

    delegator.create(newValue);
}
catch (GenericEntityException e)
{
    return ServiceUtil.returnError("Could update note data (write failure): " + e.getMessage());
}
Map result = ServiceUtil.returnSuccess();

result.put("noteId", noteId);
return result;
}

```

2. Пример создания нового экземпляра элемента данных типа Visit в службе makeALotOfVisits компонента common. В методе makeValue создается пустой экземпляр, а значения атрибутов устанавливаются путем нескольких вызовов метода set. Обратите внимание, как атрибут visitId инициализируется уникальными последовательными значениями с помощью последовательностей (Sequence).

```

public static Map makeALotOfVisits(DispatchContext dctx, Map context)
{
    GenericDelegator delegator = dctx.getDelegator();

    int count = ((Integer) context.get("count")).intValue();
    for (int i = 0; i < count; i++ )
    (
        GenericValue v = delegator.makeValue("Visit", null);
        String seqId = delegator.getNextSeqId("Visit").toString();

        v.set("visitId", seqId);
        v.set("userCreated", "N");
        v.set("sessionId", "NA-" + seqId);
        v.set("serverIpAddress", "127.0.0.1");
        v.set("serverHostName", "localhost");
        v.set("webappName", "webtools");
        v.set("initialLocale", "en_US");
        v.set("initialRequest", "http://localhost:8080/webtools/control/main");
        v.set("initialReferrer", "http://localhost:8080/webtools/control/main");
        v.set("initialUserAgent", "Mozilla/5.0 (Macintosh; U; PPC Mac OS X;
en-us) AppleWebKit/124 (KHTML, like Gecko) Safari/125.1");
    )
}

```

```
v.set("clientIpAddress", "127.0.0.1");  
v.set("clientHostName", "localhost");  
v.set("fromDate", UtilDateTime.nowTimestamp());
```

```
try  
{  
    delegator.create(v);  
}  
catch (GenericEntityException e)  
{  
    Debug.logError(e, module);  
}  
}  
  
return ServiceUtil.returnSuccess();  
}
```

3. Пример поиска по первичному ключу и модификации значений атрибутов у найденного экземпляра элемента данных типа JobSandBox в службе cancel-ScheduledJob компонента service.

```
public static Map cancelJob(DispatchContext dctx, Map context)  
{  
    GenericDelegator delegator = dctx.getDelegator();  
    Security security = dctx.getSecurity();  
    GenericValue userLogin = (GenericValue) context.get("userLogin");  
    Locale locale = getLocale(context);  
  
    if (!security.hasPermission("SERVICE_INVOKE_ANY", userLogin))  
    {  
        String errMsg = UtilProperties.getMessage(ServiceUtil.resource,  
"serviceUtil.no_permission_to_run", locale) + ".";  
        return ServiceUtil.returnError(errMsg);  
    }  
  
    String jobId = (String) context.get("jobId");  
    Map fields = UtilMisc.toMap("jobId", jobId);  
  
    GenericValue job = null;  
    try  
    {
```

```

job = delegator.findByPrimaryKey("JobSandbox", fields);
if (job != null)
{
    job.set("cancelDateTime", UtilDateTime.nowTimestamp());
    job.set("statusId", "SERVICE_CANCELLED");
    job.store();
}
}
catch (GenericEntityException e)
{
    Debug.logError(e, module);
    String errMsg = UtilProperties.getMessage(ServiceUtil.resource,
"serviceUtil.unable_to_cancel_job", locale) + " : " + fields;

    return ServiceUtil.returnError(errMsg);
}

```

```

Timestamp cancelDate = job.getTimestamp("cancelDateTime");
if (cancelDate != null)
{
    Map result = ServiceUtil.returnSuccess();
    result.put("cancelDateTime", cancelDate);
    return result;
}

```

4. Пример поиска по значениям неключевых атрибутов экземпляров элемента данных типа Geo в классе `CommonWorkers.java` компонента `common`. Найденные экземпляры сортируются в соответствии со значением атрибута `geoName`.

```

public static List getStateList(GenericDelegator delegator)
{
    List geoList = new ArrayList();
    Map findMap = UtilMisc.toMap("geoTypeId", "STATE");
    List sortList = UtilMisc.toList("geoName");

    try
    {
        geoList = delegator.findByAndCache("Geo", findMap, sortList);
    }
    catch (GenericEntityException e)

```

```
{
    Debug.LogError(e, "Cannot lookup Geo", module);
}
```

```
return geoList;
```

```
}
```

5. Пример поиска по составному условию с последующим удалением найденных экземпляров элемента данных типа JobSandBox в службе purgeOldJobs компонента service.

```
public static Map purgeOldJobs(DispatchContext dctx, Map context)
```

```
{
    String sendPool = ServiceConfigUtil.getSendPool();
```

```
int daysToKeep = ServiceConfigUtil.getPurgeJobDays();
```

```
GenericDelegator delegator = dctx.getDelegator();
```

```
Timestamp now = UtilDateTime.nowTimestamp();
```

```
Calendar cal = Calendar.getInstance();
```

```
cal.setTimeInMillis(now.getTime());
```

```
cal.add(Calendar.DAY_OF_YEAR, daysToKeep * -1);
```

```
Timestamp purgeTime = new Timestamp(cal.getTimeInMillis());
```

```
// create the conditions to query
```

```
EntityCondition pool = new EntityExpr("poolId", EntityOperator.EQUALS, sendPool);
```

```
List finExp = UtilMisc.toList(new EntityExpr("finishDateTime",
```

```
EntityOperator.NOT_EQUAL, null));
```

```
finExp.add(new EntityExpr("finishDateTime", EntityOperator.LESS_THAN, purgeTime));
```

```
List canExp = UtilMisc.toList(new EntityExpr("cancelDateTime",
```

```
EntityOperator.NOT_EQUAL, null));
```

```
canExp.add(new EntityExpr("cancelDateTime", EntityOperator.LESS_THAN, purgeTime));
```

```
EntityCondition cancelled = new EntityConditionList(canExp, EntityOperator.AND);
```

```
EntityCondition finished = new EntityConditionList(finExp, EntityOperator.AND);
```

```
EntityCondition done = new EntityConditionList(UtilMisc.toList(cancelled, finished),
```

```
EntityOperator.OR);
```

```
EntityCondition main = new EntityConditionList(UtilMisc.toList(done, pool),
```

```
EntityOperator.AND);
```

```
// lookup the jobs
```

```
List foundJobs = null;
```

```
try
```

```
{
    foundJobs = delegator.findByCondition("JobSandBox", main, null, null);
}
```

```
catch (GenericEntityException e)
```

```
{
    Debug.LogError(e, "Cannot get jobs to purge");
    return ServiceUtil.returnError(e.getMessage());
}
```

```
if (foundJobs != null && foundJobs.size() > 0)
```

```
{
```

```

Iterator i = foundJobs.iterator();
while (i.hasNext())
{
    GenericValue job = (GenericValue) i.next();
    try
    {
        job.remove();
    }
    catch (GenericEntityException e)
    {
        Debug.logError(e, "Unable to remove job : " + job, module);
    }
}
)
)

return ServiceUtil.returnSuccess();
}

private boolean hasBom(GenericValue product, Date inDate) throws
GenericEntityException
(
    List children = product.getRelatedByAnd("MainProductAssoc",
        UtilMisc.toMap("productAssocTypeId", bomTypeId));
    children = EntityUtil.filterByDate(children, inDate);
    return (children != null && children.size() > 0);
}

```

## ***8.8. Пример использования ЦУЭД в Web-приложении biminfo***

В нашем собственном модуле biminfo мы можем намного расширить функциональность Web-приложения, если воспользуемся возможностями ЦУЭД. В качестве учебного примера реализуем подписку на учебные курсы студентами и выдачу сводки по подписке на курсы. В возможности Web-приложения будет входить:

- 1) регистрация студента на учебный курс;
- 2) просмотр списка курсов, на которые студент зарегистрировался;
- 3) удаление элементов из списка зарегистрированных курсов.

Внесем изменения в главный экран приложения, добавив пункт меню «Your Courses Subscription» и для каждого предлагаемого курса ссылку «subscribe...» (рис. 8.13).

Диаграмма переходов в нашем Web-приложении приобретет следующий вид (рис. 8.14).

# BimApplication

Main Schedule Advisor You Courses Subscription

## Proposed Courses List

Graphics Design

View Description...

subscribe...

Software Engineering

View Description...

subscribe...

General Studies in Political Economics

View Description...

subscribe...

# Welco of the H

Рис. 8.13. Новые элементы интерфейса в Web-приложении biminfo

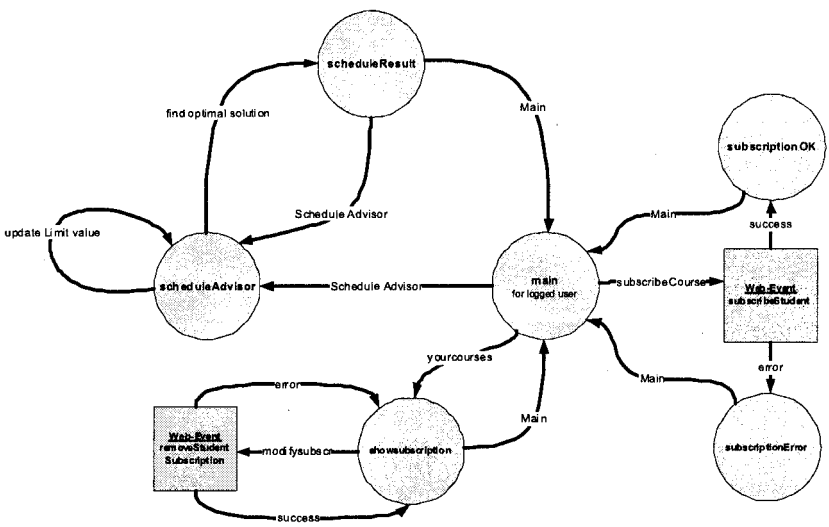


Рис. 8.14. Обновленная диаграмма переходов для Web-приложения biminfo

Соответствующие изменения в файле:

`%OFBIZ_HOME%\specialized\bim-info\webapp\biminfo\WEBINF\controller.xml`

приведены на рис. 8.15.

Добавления в файлы с определениями экранов:

`%OFBIZ_HOME%\specialized\biminfo\widget\MainScreens.xml`

и форм:

`%OFBIZ_HOME%\specialized\biminfo\widget\forms.xml`

несложные и похожи на сделанные в предыдущих главах.

Обратимся к организации данных в измененном Web-приложении biminfo. Для хранения информации о подписке на курсы нам потребуется новый тип элемента данных, который используется для связи экземпляров элементов данных Content и Party. Назовем новый тип элемента

```
<!-- Request Mappings -->
....
<request-map uri="subscribeCourse">
  <security https="true" auth="true"/>
  <event type="java" invoke="subscribeStudent" path="org.hse.bim.web.bimEvents"/>
  <response name="success" type="view" value="subscriptionOK"/>
  <response name="error" type="view" value="subscriptionError"/>
</request-map>
<request-map uri="yourcourses">
  <security https="true" auth="true"/>
  <response name="success" type="view" value="showssubscription"/>
</request-map>
<request-map uri="modifysubscr">
  <security https="true" auth="true"/>
  <event type="service-multi" invoke="removeStudentSubscription"/>
  <response name="success" type="view" value="showssubscription"/>
</request-map>
...
<!-- View Mappings -->
...
<view-map name="subscriptionOK" type="screen"
page="component://biminfo/widget/MainScreens.xml#subscriptionOK"/>
<view-map name="subscriptionError" type="screen"
page="component://biminfo/widget/MainScreens.xml#subscriptionError"/>
<view-map name="showssubscription" type="screen"
page="component://biminfo/widget/MainScreens.xml#showssubscription"/>
...
```

**Рис. 8.15.** Фрагмент файла controller.xml, где показаны новые обрабатываемые запросы и представления



```

<?xml version="1.0" encoding="UTF-8"?>
<entitymodel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/entitymodel.xsd">
  <!-- ===== -->
  <!-- Defaults ===== -->
  <!-- ===== -->
  <title>Entity of an Open For Business Project Component</title>
  <description>BIM Component</description>
  <copyright>Copyright (c) 2005 </copyright>
  <author>None</author>
  <version>1.0</version>
  <!-- ===== -->
  <!-- Data Model ===== -->
  <!-- The modules in this file are as follows: -->
  <!-- - org.hse.bim -->
  <!-- ===== -->
  <!-- org.hse.bim -->
  <!-- ===== -->
  <entity entity-name="CourseEnrollment" package-name="org.hse.bim" title="Course
Enrollment Entity">
    <field name="contentId" type="id-ne"/>
    <field name="partyId" type="id-ne"/>
    <prim-key field="contentId"/>
    <prim-key field="partyId"/>
    <relation type="one" fk-name="ENRLMNT_RL_CNTNT" rel-entity-name="Content">
      <key-map field-name="contentId"/>
    </relation>
    <relation type="one-nofk" rel-entity-name="Party">
      <key-map field-name="partyId"/>
    </relation>
  </entity>
</entitymodel>

```

**Рис. 8.16.** Файл

%OFBIZ\_HOME%\specialized\biminfo\entitydef\entitymodel.xml  
с определением нового типа элементов данных CourseEnrollment

```

<?xml version="1.0" encoding="UTF-8"?>
<entitygroup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/entitygroup.xsd">
  <!-- =====org.hse.bim =====>
  <entity-group group="org.ofbiz" entity="CourseEnrollment"/>
</entitygroup>

```

**Рис. 8.17.** Файл

%OFBIZ\_HOME%\specialized\biminfo\entitydef\entitygroup.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="biminfo" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/ofbiz-
component.xsd">
  <!-- define resource loaders; most common is to use the component resource loader -->
  <resource-loader name="main" type="component"/>
  <!-- load single or multiple external libraries -->
  <classpath type="jar" location="build/lib/*"/>
  <!-- place the config directory on the classpath to access configuration files -->
  <classpath type="dir" location="config"/>
  <classpath type="dir" location="script"/>
  <!-- entity resources: model(s), eca(s) and group definitions -->
  <entity-resource type="model" reader-name="main" loader="main"
location="entitydef/entitymodel.xml"/>
  <entity-resource type="group" reader-name="main" loader="main"
location="entitydef/entitygroup.xml"/>
  <service-resource type="model" loader="main" location="servicedef/services.xml"/>
  <!-- web applications; will be mounted when using the embedded Jetty container -->
  <webapp name="biminfo" title="BIM INFO" server="default-server"
location="webapp/biminfo" mount-point="/biminfo"/>
</ofbiz-component>

```

**Рис. 8.18.** Файл  
%OFBIZ\_HOME%\specialized\biminfo\ofbiz-component.xml

```

1. // Subscribe a student to specific course (new Course Enrollment Entities are used )
2. public static String subscribeStudent(HttpServletRequest request,
3.     HttpServletResponse response)
4.     throws GeneralException
5.     {
6.     GenericDelegator delegator = (GenericDelegator) request.getAttribute("delegator");
7.     Security security = (Security) request.getAttribute("security");
8.     HttpSession session = request.getSession();
9.     GenericValue userLogin = (GenericValue) session.getAttribute("userLogin");
10.    GenericValue party = null;
11.    GenericValue enrollment = null;
12.    boolean hasEnrollment;
13.    String userLoginId = userLogin.getString("userLoginId");
14.    Try
15.    {
16.        party = (userLogin == null)?null: userLogin.getRelatedOne("Party");
17.    }
18.    catch (GenericEntityException gee)
19.    {
20.        Debug.logWarning(gee, module);
21.        return "error";
22.    }
23.    String partyId = party.getString("partyId");
24.    if(UtilValidate.isEmpty(partyId))

```

**Рис. 8.19.** Реализация Web-event

```

25.     {
26.         request.setAttribute("_ERROR_MESSAGE_", "Student partyId is empty");
27.         return "error";
28.     }
29.     // extract parameters from the HTTP request
30.     Map paramMap = (Map) UtilHttp.getParameterMap(request);
31.     String contentId = (String) paramMap.get("contentId");
32.     if(UtilValidate.isEmpty(contentId))
33.         { request.setAttribute("_ERROR_MESSAGE_", "Course ContentId parameter is empty");
34.
35.             return "error";
36.         }
37.     //check previous enrollments
38.     hasEnrollment = false;
39.     try
40.     {
41.         List enrollmentList = delegator.findByAndCache("Course Enrollment",
42.                                                         UtilMisc.toMap("partyId", partyId, "contentId", contentId));
43.         if (enrollmentList.size() > 0)
44.         {
45.             hasEnrollment = true;
46.         }
47.     } catch (GenericEntityException e)
48.     {
49.         Debug.logWarning(e, module); return "error";
50.     }
51.     if (! hasEnrollment)
52.         { // create new instance of CourseEnrollment
53.             Try
54.             {
55.                 enrollment = delegator.makeValue("Course Enrollment",
56. null);
57.                 enrollment.set("contentId", contentId);
58.                 enrollment.set("partyId", partyId);
59.                 enrollment.create();
60.             }
61.             catch (GenericEntityException e)
62.             {
63.                 Debug.logWarning(e, module); return "error";
64.             }
65.         }
66.     }

```

**Рис. 8.19.** Реализация Web-event (окончание)

данных Course-Enrollment. Он будет иметь два атрибута: contentId и partyId. Все необходимые сведения по структуре этого элемента данных занесем (рис. 8.16) в файл:

**%OFBIZ\_HOME%\specialized\biminfo\entitydef\entitymodel.xml.**

Не забудем зарегистрировать созданный элемент данных в группе и внести изменения в главный конфигураци-

онный файл компонента ofbiz-component.xml (рис. 8.17, 8.18).

Теперь наступает черед изучения разработанных алгоритмов по модификации элементов данных (рис. 8.19). Создание нового экземпляра CourseEnrollment выполняется в классе org.hse.bim.web.bimEvents.java.

```
1.      /**
2.       * removeStudentSubscription
3.       *
4.       * removes CourseEnrollment entities by partyId and courseId keys
5.       *
6.       */
7.      public static Map removeStudentSubscription(DispatchContext dctx, Map context)
8.      {
9.
10.         LocalDispatcher dispatcher = dctx.getDispatcher();
11.         GenericDelegator delegator = dctx.getDelegator();
12.
13.         GenericValue userLogin = (GenericValue)context.get("userLogin");
14.         String userLoginPartyId = userLogin.getString("partyId");
15.         Map results = new HashMap();
16.
17.         String courseContentId = (String)context.get("contentId");
18.         String partyId = (String)context.get("partyId");
19.
20.         if (UtilValidate.isEmpty(courseContentId) || UtilValidate.isEmpty(partyId))
21.             return results;
22.
23.         Try
24.         {
25.             delegator.removeByAnd("CourseEnrollment",
26.                 UtilMisc.toMap("partyId", partyId, "contentId", courseContentId),
27. true);
28.         } catch (GenericEntityException e)
29.         {
30.             Debug.logError(e, "Cannot get enrollments");
31.             return ServiceUtil.returnError(e.getMessage());
32.         }
33.         return results;
34.     }
```

Рис. 8.20. Реализация службы

Для удаления информации о подписке на определенную курс используется служба (рис. 8.20). Ее алгоритм реализован в классе `org.hse.bim.schedule.ScheduleServices.java`.

## 8.9. Контрольные задания к главе 8

1. Создать пользователя с особыми правами (преподавателя). Дополнить интерфейс Web-приложения BIMINFO новым экраном, который разрешено отображать только для такого пользователя. На этом экране вывести списки зарегистрированных студентов для всех курсов.

2. Создать новый тип элемента данных `CourseTutor`, включив туда ссылку на преподавателя (`partyId`) и ссылку на описание курса (`contentId`). Разработать экраны и нужные алгоритмы, чтобы пользователь `admin` мог устанавливать и удалять связи между курсами и преподавателями.

3. Модифицировать задание № 1, чтобы каждому преподавателю отображалась сводка только по его курсам.

4. Создать набор правил `Entity Event-Condition-Action`, чтобы преподавателю отправлялось почтовое сообщение при изменении связи между курсом и преподавателем.

5. N&N Forever II: дополнить реализацию прототипа информационной системы производства корпорации «Рога и Копыта». Все данные о структуре производства и запасах на складах должны находиться под управлением ЦУЭД. Максимально нужно использовать существующие типы элементов данных (`Facility`, `Manufacturing` и др.) — создание своих типов элементов данных крайне нежелательно и допускается лишь как редкое исключение.

6. В примерах по подписке на курсы (классы `bimEvents.java` и `ScheduleServices.java`) замените поиск с помощью метода `findByAndCache` и удаление с помощью метода `removeByAnd` на соответствующие методы с использованием первичного ключа (`findByPrimaryKey` и `removeByPrimaryKey`). Оцените разницу в производительности.

## 8.10. Приложение к главе 8

ЦУЭД предоставляет разработчику возможность определения автоматической реакции на события, происходя-

щие с экземплярами различных типов элементов данных во время работы. Такой механизм носит название *Entity Event-Condition-Action* (ЕЕСА). Определение автоматической реакции создается в отдельном XML-файле, который должен быть построен в соответствии с XML-схемой

`%OFBIZ_HOME%\framework\entity\dtd\entity-eca.xsd`.

На рис. 8.21 представлен пример такого определения.

Проверка может осуществляться при выполнении таких операций над элементами данных (разрешенные значения атрибута operation):

- ◆ create;
- ◆ store;
- ◆ remove;

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-eca xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/entity-eca.xsd">
  <!-- Inventory Issue ECAs for Immediately Fulfilled Orders -->
  <eca entity="OrderHeader" operation="create-store" event="return">
    <condition field-name="statusId" operator="equals" value="ORDER_COMPLETED"/>
    <condition field-name="needsInventoryIssuance" operator="equals" value="Y"/>
    <action service="issueImmediatelyFulfilledOrder" mode="sync"/>
  </eca>
  <!-- Inventory Receive ECAs for Immediately Received Returns -->
  <eca entity="ReturnHeader" operation="create-store" event="return">
    <condition field-name="statusId" operator="equals" value="RETURN_ACCEPTED"/>
    <condition field-name="needsInventoryReceive" operator="equals" value="Y"/>
    <action service="quickReceiveReturn" mode="sync"/>
  </eca>
  <eca entity="OrderItem" operation="create-store" event="return">
    <action service="checkCreateOrderRequirement" mode="sync"/>
  </eca>
</entity-eca>
```

Рис. 8.21. Определение реакции ЕЕСА

- ◆ find;
- ◆ create-store;
- ◆ create-remove;
- ◆ store-remove;
- ◆ create-store-remove;
- ◆ any.

```

<?xml version="1.0" encoding="UTF-8"?>
<ofbiz-component name="product" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/ofbiz-
component.xsd">
  <resource-loader name="main" type="component"/>
  <classpath type="dir" location="config"/>
  <classpath type="dir" location="script"/>
  <classpath type="jar" location="build/lib/**/*>
  <entity-resource type="model" reader-name="main" loader="main"
location="entitydef/entitymodel.xml"/>
  <entity-resource type="group" reader-name="main" loader="main"
location="entitydef/entitygroup.xml"/>
  <entity-resource type="eca" reader-name="main" loader="main"
location="entitydef/eecas.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/ProductTypeData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/ShipmentTypeData.xml"/>
  <entity-resource type="data" reader-name="seed" loader="main"
location="data/ProductSecurityData.xml"/>
  <service-resource type="model" loader="main" location="servicedef/services.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_facility.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_feature.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_maint.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_picklist.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_pricepromo.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_shipment_ups.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_shipment_usps.xml"/>
  <service-resource type="model" loader="main"
location="servicedef/services_shipment.xml"/>
  <service-resource type="model" loader="main" location="servicedef/services_store.xml"/>
  <service-resource type="model" loader="main" location="servicedef/services_view.xml"/>
  <service-resource type="eca" loader="main" location="servicedef/secas.xml"/>
  <service-resource type="eca" loader="main" location="servicedef/secas_shipment.xml"/>
  <service-resource type="group" loader="main"
location="servicedef/groups_shipment.xml"/>
  <webapp name="catalog" title="Catalog" server="default-server"
location="webapp/catalog" base-permission="CATALOG" mount-point="/catalog"/>
  <webapp name="facility" title="Facility" server="default-server" location="webapp/facility"
base-permission="FACILITY" mount-point="/facility"/>
</ofbiz-component>

```

**Рис. 8.22.** Подключение файла с определением реакции ЕЕСА в основном конфигурационном файле ofbiz-component.xml

События выбираются из такого списка (разрешенные значения атрибута event):

- ◆ validate;
- ◆ run;
- ◆ return;
- ◆ cache-check;
- ◆ cache-put;
- ◆ cache-clear.

В ходе проверки проводится сравнение атрибутов элемента данных с определенным значением. Если все проверки были выполнены успешно, то наступает реакция на событие. В качестве реакции разработчик может указать имя произвольной службы. Все созданные файлы с определениями ЕЕСА должны быть указаны в файле конфигурации ofbiz-component.xml (рис. 8.22).



## **Глава 9**

### **Автоматизации бизнес-процессов в системе OFBIZ. Принципы разработки приложений**

Методы разработки графического интерфейса, служб и элементов данных составляют практически полный набор инструментов для создания полнофункциональных приложений в системе OFBIZ. В этой завершающей главе по системе OFBIZ мы рассмотрим дополнительные возможности, которые предоставляет разработчикам система, а также познакомимся с рекомендациями по разработке, которым создатели системы советуют следовать в ходе проектирования и реализации.

#### ***9.1. Автоматизация бизнес-процессов с помощью OFBIZ Workflow Engine***

В состав системы OFBIZ включен особый компонент Workflow Engine (%OFBIZ\_HOME%\framework\workflow, Web-приложение «Workflow»), предназначенный для автоматизированного выполнения бизнес-процессов. Этот компонент реализован в соответствии со спецификациями и стандартами WfMC и OMG. Следовательно, для определения структуры бизнес-процессов, правил переходов и используемых данных в нем применяются уже известные вам понятия (функция, участник и т.п.), а также форматы файлов с описаниями бизнес-процессов, например, язык XPDЛ.

Отметим, что реализация Workflow Engine в системе OFBIZ была одной из первых свободно распространяемых реализаций стандарта WfMC на языке Java и послужила основой для независимого и хорошо вам знакомого проекта ObjectWeb Shark. Некоторое время оригинальный OFBIZ

Workflow Engine эволюционировал независимо, а потом было принято решение провести интеграцию ObjectWeb Shark в состав системы OFBIZ. Однако интеграция завершена лишь частично, поэтому, по сути, в системе OFBIZ можно параллельно использовать для управления бизнес-процессами две возможности: оригинальную реализацию OFBIZ Workflow Engine и частично проинтегрированный Shark. Интеграция этого продукта ведется в компоненте %OFBIZ\_HOME%\framework\shark (Web-приложение «Shark»), и после ее завершения Shark станет основным средством управления бизнес-процессами в системе OFBIZ. В этой главе мы отдельно опишем возможности оригинального Workflow Engine и возможности текущей интеграции Shark с разъяснениями в тех местах, где имеется разница в использовании.

Обе реализации Workflow Engine тесно связаны с основными составляющими системы OFBIZ — ЦУС и ЦУЭД. В определении функций бизнес-процесса можно использовать службы, да и сами бизнес-процессы, как вы увидите позднее, могут выглядеть для пользователей системы OFBIZ как традиционные службы. В работе Workflow Engine активно используются элементы данных, определенные в файле entitymodel\_workflow.XML entitymode\_workeffort, для долговременного хранения описаний бизнес-процессов и детальной информации о выполнении активных экземпляров процессов. Таким образом, все изменения в данных бизнес-процессов, информация о состоянии экземпляров защищены от прерывания работы приложения или системы в целом. И если приложение случайно выходит из строя, то Workflow Engine продолжит после перезапуска свою работу с последней выполняющейся функции, прочитав всю необходимую информацию из ЦУЭД.

## ***9.2. Оригинальный OFBIZ Workflow Engine***

### ***9.2.1. Реализованные типы функций в описании бизнес-процессов***

В оригинальной реализации OFBIZ Workflow Engine поддерживается большинство типов функций бизнес-процессов по стандартам WfMC, включая NO, ROUTE, SUB-

FLOW. Функции могут исполняться в ручном и автоматическом режиме, причем для автоматического исполнения может использоваться только один тип приложения — TOOL:Procedure. Тип TOOL:Application в настоящее время не поддерживается. Также не поддерживаются функции типа LOOPS.

### 9.2.2. Особые расширения в XPDL-описании

С целью полной интеграции с такими специфическими возможностями системы OFBIZ, как службы, OFBIZ Workflow Engine воспринимает в XPDL-файле ряд расширений. Большинство расширений реализовано с помощью стандартного механизма ExtendedAttributes, с которым вы уже должны быть знакомы.

В определении фактических параметров при вызове приложений Work-flow разработчик системы OFBIZ может указать дополнительные сведения.

◆ **Выражения (Expressions):** Можно использовать выражения внутри ActualParameters, например:

```
<ActualParameter>expr:orderNumber=orderId</  
ActualParameter>
```

will map the context attribute *orderId* to a buffered variable named *orderNumber* then if you did:

```
<ActualParameter>orderNumber</ActualParameter>
```

the parameter *orderNumber* would be passed to the service with the value of *orderId*.

◆ **Идентификатор задания (WorkEffort ID):** Можно использовать *workEffortId* как ActualParameter, что будет отражено в primary key.

### 9.2.3. Использование workflow Engine

Для того чтобы использовать возможности оригинального OFBIZ Work-flow, Engine необходимо, прежде всего, иметь описание бизнес-процесса в формате XPDL. Это описание должно быть импортировано в систему OFBIZ, где преобразуется в совокупность элементов данных под управлением ЦУЭД. В оригинальной реализации OFBIZ

Framework Web Tools

**Read XPDL File**

This page is used to read and import XPDL files into the workflow process repository

XPDL Filename or URL:  Is URL?:

Import/Update to DB?:

**The following occurred:**

- Could not find file/URL: null

No toBeStored read

**Рис. 9.1.** Загрузка описания бизнес-процесса из XPDL-файла в оригинальный OFBIZ Workflow Engine

Workflow Engine импорт определений происходит в Web-приложении Webtools (ссылка «Read XPDL File» на главной странице, рис. 9.1).

После успешного импорта определений бизнес-процесс становится доступным для запуска. Типичным способом использования бизнес-процессов в системе OFBIZ является их определение в виде службы. При этом единственным отличием определения бизнес-процесса в качестве службы от ранее рассмотренных нами примеров является использование другого значения для атрибута «engine» XML-элемента «service». Если служба реализована как описание бизнес-процесса, то требуется, чтобы значение атрибута engine было 'workflow'. После редактирования файла с интерфейсами служб и очистки кеша в системе OFBIZ запуск бизнес-процесса ничем не отличается от запуска служб (например, можно запустить бизнес-процесс как задание — Job). На рис. 9.2 приведен пример определения в виде службы бизнес-процесса по обработке заказов в компоненте order — фрагмент (рис. 9.2) файла:

`%OFBIZ_HOME%\applications\order\servicedef\services.xml).`

Соответствующее определение бизнес-процесса находится в файле:

`%OFBIZ_HOME%\applications\servicedef\ orderProcessXPDL.xml.`

```
<service name="processOrderWf" engine="workflow" location="org.ofbiz.order"
invoke="ProcessOrder">
  <description>Service for testing the workflow engine</description>
  <attribute name="orderId" type="String" mode="IN" optional="false"/>
</service>
```

**Рис. 9.2.** Определение бизнес-процесса в роли службы OFBIZ

Обратите внимание, что определение службы в качестве значения атрибута location использует значение атрибута id XML-элемента package из XPDЛ файла.

### ***9.3. Возможности проинтегрированного Workflow Engine Shark в составе системы OFBIZ***

В последних версиях системы OFBIZ (май 2006) Shark практически уже стал Workflow Engine по умолчанию<sup>42</sup>. По крайней мере, бизнес-процессы, определенные как службы OFBIZ, теперь обрабатываются именно Shark-ом<sup>43</sup>. Первое следствие этого заключается в том, что загрузка XPDЛ-файла с определениями бизнес-процессов должна теперь производиться в Web-приложении Shark на странице Repository. Воспользуемся для демонстрации работы Shark Workflow Engine иллюстративным описанием бизнес-процесса (рис. 9.3) на языке XPDЛ, располагающимся в каталоге %OFBIZ\_HOME%\framework\shark\example\ofb\_test.xpdл.

Прежде всего необходимо этот файл скопировать в каталог-репозиторий

%OFBIZ\_HOME%\framework\shark\xpdл.

После этого можно запустить Web-приложение Shark (рис. 9.4).

В этом Web-приложении по ссылке XPDЛ Repository перейдем на страницу управления описаниями бизнес-процессов и загрузим в систему OFBIZ описание ofb\_test.xpdл (рис. 9.5).

В случае успешной загрузки на странице появится таблица с перечислением загруженных в систему OFBIZ

<sup>42</sup> Однако интеграция еще не завершена — нет связи с компонентом WorkEfforts и выражения в XPDЛ-описании обрабатываются лишь частично.

<sup>43</sup> Причем можно использовать для конфигурации Shark Workflow Engine помимо средств OFBIZ стандартное административное приложение Shark Admin Tools.

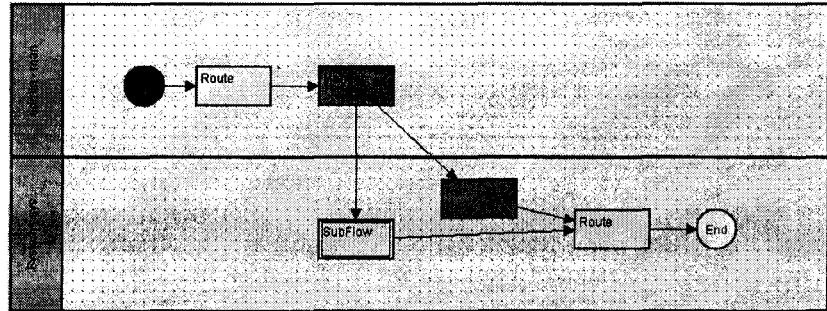


Рис. 9.3. Простой бизнес-процесс

определений. Как это делается обычно в Shark, требуемое определение нужно открыть для дальнейшего использования (ссылка open на рис. 9.6).

После этого с главной страницы Web-приложения Shark можно перейти по ссылке Process List на страницу, где создаются новые экземпляры процессов (ссылка create)

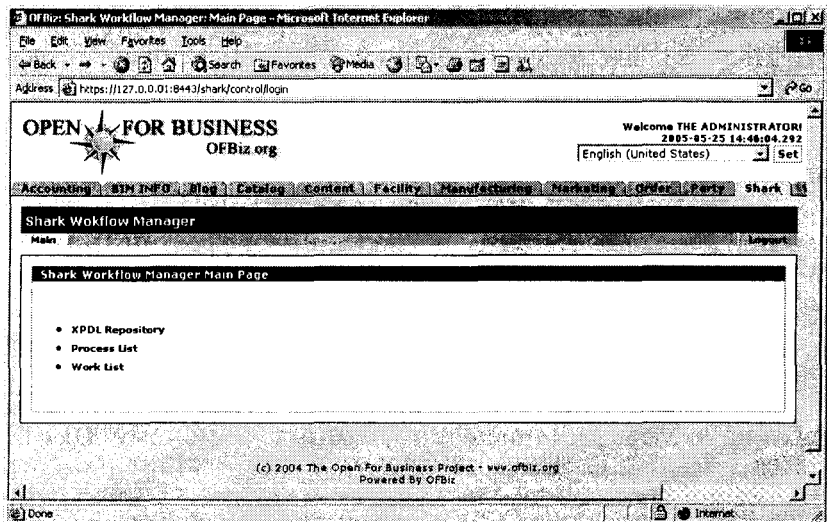


Рис. 9.4. Страница для работы с Shark в составе OFBIZ

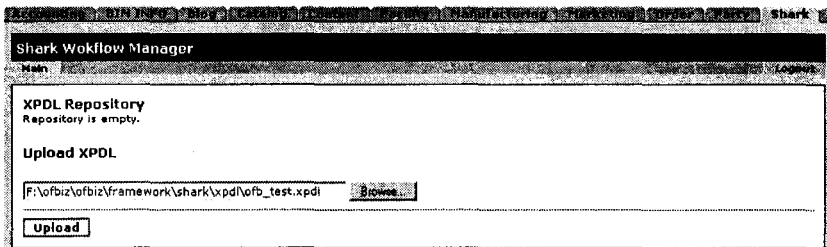


Рис. 9.5. Загрузка описаний в репозиторий Shark

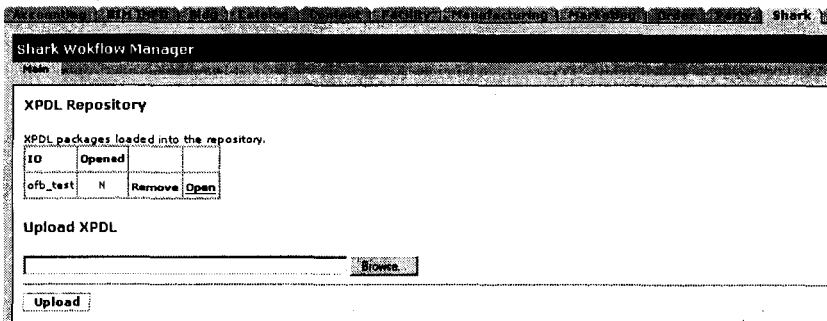


Рис. 9.6. Управление загруженными описаниями

и производится управление текущим состоянием запущенных экземпляров (ссылка View) (рис. 9.7).

Заметим, что определения, загруженные таким образом, также могут играть роль служб. Пример представления бизнес-процесса, выполняющегося в Shark, в виде службы есть в текущей версии системы OFBIZ<sup>44</sup> (рис. 9.8).

<sup>44</sup> В качестве значения атрибута invoke используется Id бизнес-процесса из описания на XPDL.

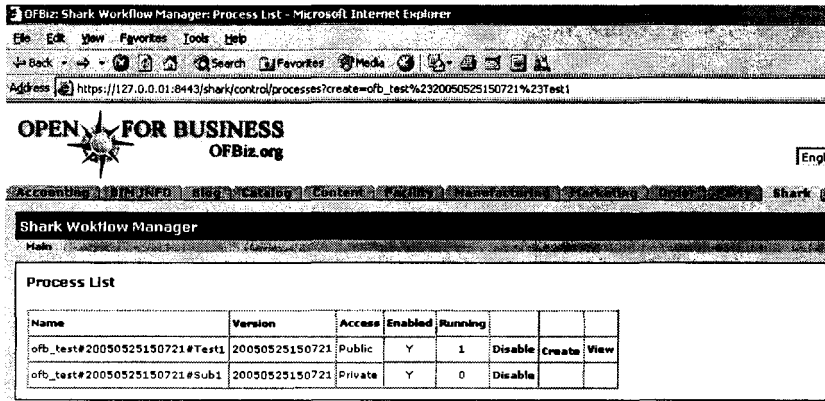


Рис. 9.7. Управление процессами в Shark

Как обычно, такие службы могут быть вызваны из Java-компонентов либо запущены через Web-приложение Webtools. В последнем случае используется уже упомянутая ранее ссылка Schedule Job.

В нашем примере при запуске процесса Test1 первая функция стартует в ручном режиме. Подтвердить ее исполнение и завершить работу можно непосредственно в Web-браузере: в состав Web-приложения Shark входит страница по управлению назначенными на исполнение функциями в ручном режиме (аналог административного приложения Shark). На эту страницу можно перейти по ссылке Work List с главной страницы Web-приложения Shark (рис. 9.4, 9.9).

```
<services xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://www.ofbiz.org/dtds/services.xsd">
  <description>OFBIZ Shark Test Services</description>
  <vendor>OFBiz</vendor>
  <version>1.0</version>
  <service name="sharkTest" engine="workflow" export="true" use-transaction="false"
location="ofb_test" invoke="Test1">
    <description>Shark Workflow Test Service</description>
  </service>
</services>
```

Рис. 9.8. Определение бизнес-процесса в роли службы OFBIZ



OFBiz: Shark Workflow Manager: Work List - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Search Favorites Media Print

Address https://127.0.0.1:8443/shark/control/worklist

**OPEN FOR BUSINESS**  
OFBiz.org

Shark

Shark Workflow Manager

Main

Work List

Resource	Name	Activity	Accepted	Priority	State	State Change	
admin	Manual	10010_10008_ofb_test_Test1	<input checked="" type="checkbox"/>	3	open.running	2005-05-25 15:13:57.044	Complete

Рис. 9.9. Управление ходом выполнения ручных функций бизнес-процесса

```

--- SVC-CONTEXT: locale => en_US
--- SVC-CONTEXT: ofb_test_App1_For1 => Sub-Flow Message!
---- SVC: WFDispatcher ----
3204454
(org.enhydra.shark.ThreadedToolAgentManager$ToolRunnerManager$ToolRunner
){ ServiceDispatcher.java:442:DEBUG} [[Sync service finished- total:30.003,si
ce last(Begin):30.003]] - 'WFDispatcher / blockingTestScv'
--- SVC-CONTEXT: locale => en_US
--- SVC-CONTEXT: ofb_test_App1_For1 => OFBiz Shark Test
---- SVC: WFDispatcher ----

```

Рис. 9.10. Результат работы службы, вызванной в ходе выполнения бизнес-процесса

После успешного завершения ручной функции происходит автоматическое выполнение функции «Run Service». При этом вызывается OFBIZ-служба `blockingTestScv`, выводя на текстовую консоль отладочные сообщения (рис. 9.10).

После полного завершения бизнес-процесса, запущенного как служба через страницу «Schedule Job», можно посмотреть статистику его выполнения по ссылке `Job List` в Web-приложении `WebTools` (рис. 9.11).

Нужно обязательно добавить, что управлять ходом выполнения бизнес-процесса можно также и с помощью стандартного административного приложения `Shark`, взаи-

Job	Pool	Run Time	Start Time	Service	Finish Time	
Run Auto-Reorders	pool	2005-05-28 04:00:00.0		runShoppingListAutoReorder		[Cancel Job]
Order Auto-Cancel	pool	2005-05-28 03:00:00.0		autoCancelOrderItems		[Cancel Job]
Re-Try Failed Auths	pool	2005-05-28 01:00:00.0		retryFailedAuths		[Cancel Job]
Purge Old Jobs	pool	2005-05-28 00:00:00.0		purgeOldJobs		[Cancel Job]
Purge Old Store Auto-Entered Promos	pool	2005-05-28 00:00:00.0		purgeOldStoreAutoPromos		[Cancel Job]
Clear Entity Sync Remove Info	pool	2005-05-28 00:00:00.0		clearSyncRemoveInfo		[Cancel Job]
Back Order Notification	pool	2005-05-28 00:00:00.0		checkInventoryAvailability		[Cancel Job]
1117180960454	pool	2005-05-27 12:02:39.993	2005-05-27 12:02:56.186	sharkTest	2005-05-27 12:03:57.643	

Рис. 9.11. Выполнение бизнес-процесса в виде задания OFBIZ

модействующего с Workflow Engine. Напомним, что это приложение запускается с помощью скрипта:

`%SHARK_HOME%\bin\runA.bat.`

В диалоговом окне административного приложения необходимо ввести такие параметры:

- ◆ `login: admin;`
- ◆ `password: ofbiz;`
- ◆ `host:` зависит от конкретной конфигурации;
- ◆ `port: 2000;`
- ◆ `Engine: Shark.`

В административном приложении можно выполнить основные операции — добавить/удалить описание бизнес-процесса, загрузить или выгрузить бизнес-процесс из репозитория, отобразить список заданий для пользователя и т.п. Причем в последнем случае использовать оригинальное административное приложение даже удобнее — есть возможность просмотра и редактирования используемых в функции переменных.

### 9.3.1. Расширенные атрибуты в XPDL-определении бизнес-процесса, используемые в Shark Workflow Engine

Те виды расширенных атрибутов, которые были описаны ранее в разделе по оригинальной реализации OFBIZ

Workflow Engine, можно использовать и при работе с Shark Workflow Engine. Кроме того, в описании бизнес-процессов, изначально предназначенных для исполнения Shark в составе OFBIZ, могут использоваться такие расширенные атрибуты:

◆ *в определении процесса (Process Properties)*

ActivitiesUserPropertyofb\_test\_UDP1 (значение в примере — message);

◆ *в определении функций (Activity Properties)*

ActivitiesUserPropertyofb\_test\_UDP1 (значение в примере — Shark Workflow Test);

◆ *в определении приложений (Application Properties)*

AppName (значение в примере — blockingTestScv).

ToolAgentClass (значение в примере — org.ofbiz.shark.tool.ServiceEngine Agent).

Точного описания новых видов расширенных атрибутов для работы Shark на сайте разработчиков OFBIZ еще нет, но похоже, что обязательными расширенными атрибутами являются лишь AppName и ToolAgentClass в свойствах приложений. О назначении этих расширенных атрибутов было рассказано в главе, посвященной Shark.

---

## **Глава 10**

### **Программный проект по разработке информационной системы**

#### *10.1. Постановка задачи*

##### *10.1.1. Преамбула*

Корпорация «Рога и Копыта», оценив результаты, полученные в ходе реализации и опытной эксплуатации прототипов, приняла решение о начале проектирования и реализации многофункциональной информационной системы в одном из своих основных подразделений, занимающихся производством и продажей косметической продукции.

С целью получения высоких результатов руководство корпорации решило провести тендер на разработку системы. В тендере принимают участие три независимых группы разработчиков. Каждой группе выдается один и тот же комплект документов, содержащий общее описание задачи и требования к функциональности ожидаемой системы. Часть требований обязательна и должна быть, безусловно, реализована. Оставшиеся требования являются предметом отдельных переговоров между корпорацией и руководителями групп. В ходе этих переговоров каждый руководитель в отдельности обговаривает подмножество дополнительных требований реализованных в информационной системе его группы и стоимость этого подмножества требований.

### **Общее описание задачи**

Информационная система будет использоваться для поддержки продаж и производства косметической продукции, производимой корпорацией «Рога и Копыта». Внешними пользователями этой системы будут являться розничные покупатели и различные мелкооптовые компании-распространители. Кроме того, информационная система будет использоваться работниками самой корпорации: менеджерами по продажам и производству, маркетологами, сотрудниками отдела обработки заказов, рекламными агентами.

Под управление информационной системы помещается информация рекламного характера (номенклатура продукции, цены, проводимые маркетинговые акции и т.п.), сведения о заказах и обработке рекламаций на продукцию, оперативная и аналитическая информация по производству продукции и доставке ее потребителям, основная бухгалтерская отчетность.

### **Список требований**

#### *Обязательные функциональные требования*

1. В системе должен быть портал Интернет-торговли для розничных покупателей (аналог приложения e-commerce системы OFBIZ).

2. В системе должен быть создан и наполнен информацией каталог продукции. Каталог должен позволять изменять сведения по номенклатуре товаров, о ценах (с учетом скидок, распродаж и т.п.).

3. В системе должны быть реализованы автоматизированное выполнение и контроль исполнения основных бизнес-процессов по обработке заказов, приобретению материалов, производства продукции и доставки потребителю.

#### *Обязательные требования на методы реализации*

1. Система должна быть создана средствами OFBIZ.

2. Подсистема управления каталогами должна быть основана на стандартном приложении OFBIZ Product.

## Роли участников бизнес-процесса

№	Задачи в проекте	Сложность задачи (1—10)	руководитель	технический администратор	системный архитектор	бизнес-аналитик	дизайнер
1	2	3	4	5	6	7	8
1	Взаимодействие с другими командами и заказчиком	10	100				
2	Определение и реализация бизнес-процессов на XPD L, интеграция определений в OFBIZ	7			5	95	
3	Определение конкретных заданий и сроков их исполнения, проверка исполнения заданий	9	90		10		
4	Организация внутренних митингов и совместных инспекций кода	8		40	30	30	
5	Программирование по заданным спецификациям Служб OFBIZ, скриптов, внешних Java-программ	6	20	10	50	10	10
6	Протоколирование изменений в файлах и каталогах проекта	4		50		50	
7	Разработка документации по порядку работы в портале Интернет-торговли	8		50	20	25	5
8	Разработка документации по порядку работы в стандартных приложениях OFBIZ	9	10	50	5	35	
9	Разработка единого стилевого оформления портала, создание графических изображений, CSS	7					100
10	Разработка конфигурационных файлов модуля и OFBIZ (например, controller.xml)	5			30		70
11	Разработка общей структуры системы и интерфейсов между компонентами	9	10		90		

1	2	3	4	5	6	7	8
12	Разработка регламента работы участникам бизнес-процессов (формат и назначение почтовых сообщений)	4		80		20	
13	Создание форм, экранов и FTL-шаблонов	8			20		80
14	Тестирование системы	5	20	20	20	20	20
15	Исправление ошибок в реализации и документации	8	10	30	30	15	15

3. Портал Интернет-торговли должен быть реализован как самостоятельный модуль системы OFBIZ с оригинальным и запоминающимся дизайном, отличным от стандартного дизайна, принятого в системе OFBIZ.

4. Управление ходом выполнения функций бизнес-процесса должно проходить с использованием механизмов OFBIZ Shark Workflow и Work-Effort.

### *Дополнительные требования*

1. Функции бизнес-процесса должны быть реализованы с использованием возможности МСА-Служб — по почте исполнителю функции посылаются входные данные, а он посылает письмо в определенном формате обратно, указывая результат выполнения функции.

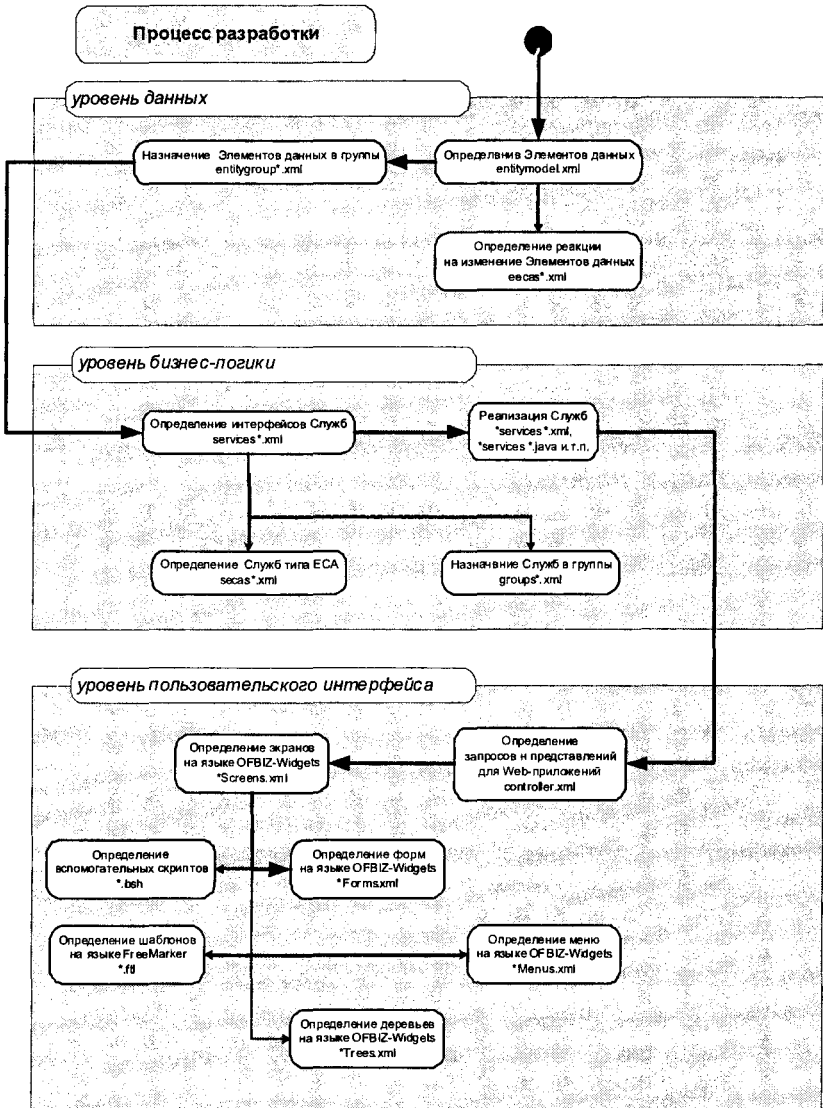
2. Должны быть созданы самостоятельные Java-приложения, предназначенные для работы основных категорий служащих (прием заказа, работа с поставщиками и т.д.).

3. В состав системы должна входить автоматизированная служба отдела кадров (Human Resources).

### *10.1.3. Организация разработки и роли участников*

В составе каждой группы находится не менее пяти участников, распределенных по следующим ролям:

- 1) руководитель проекта;
- 2) технический администратор;
- 3) системный архитектор;



**Рис. 10.1.** Предлагаемый процесс разработки информационной системы в OFBIZ



4) бизнес-аналитик;

5) дизайнер.

Рекомендуется каждой роли назначить выполнение задачи проекта в соответствии с табл. 10.1.

При необходимости состав группы может быть усилен и доведен до 7—8 человек.

## ***10.2. Общие рекомендации по принципам разработки в системе OFBIZ***

Разработчики системы OFBIZ предлагают в процессе создания собственных приложений следовать пути, определенному на рис. 10.1.

Общие советы по сокращению трудоемкости создаваемых решений можно найти в документах:

◆ «Best Pracicies Guide»

<http://www.ofbiz.org/best-practices.html>;

◆ «OFBiz Basic Production Setup Guide»

<http://www.undersunconsulting.com/static/OFBizBasicProductionSetup.pdf>.

## Библиографический список

1. 21-я Международная конференция по сетям Петри. Petri Nets. Introductory tutorial. [www.daimi.au.dk/PetriNets/introductions/pn2000\\_introtut.pdf](http://www.daimi.au.dk/PetriNets/introductions/pn2000_introtut.pdf).
2. An introduction to Petri nets. <http://viking.gmu.edu/http/syst511/vg511/AppC.html>.
3. Фиошин М. Основы р-исчисления. <http://progr.tsi.lv/research/picalc.pdf>.
4. High-level Petri Nets. Международный стандарт ISO/IEC 15909, версия — [www.informatik.hu-berlin.de/top/PNX/pnstd-4.7.1.pdf](http://www.informatik.hu-berlin.de/top/PNX/pnstd-4.7.1.pdf).
5. Van der Aalst, Hofstede A. H. M. YAWL: Yet Another Workflow Language. [www.citi.qut.edu.au/pubs/technical/yawlevtech.pdf](http://www.citi.qut.edu.au/pubs/technical/yawlevtech.pdf).
6. Van der Aalst W. M. P. Pi calculus versus Petri nets: Let us eat «humble pie» rather than further inflate the «Pi hype». [www.tm.tue.nl/staff/wvdaalst/pi-hype.pdf](http://www.tm.tue.nl/staff/wvdaalst/pi-hype.pdf).
7. Бабкин Э.А. Разработка информационных систем ERP класса / Э.А. Бабкин, О.Р. Козырев, О.Е. Полухина. Н. Новгород: НГТУ, 2006. 266 с.
8. Silverston L. The Data Model ResourceBook / L.Silverston. N.Y: John Wiley, 2001.
9. Смирнова Г. Проектирование экономических информационных систем / Г.Смирнова, А.Сорокин, Ю.Тельнов. М.: Финансы и статистика, 2001. 512 с.
10. Голенищев Э. Информационное обеспечение систем управления / Э. Голенищев, И. Клименко. М.: Феникс, 2002. 352 с.
11. О'Лири Д. ERP системы: выбор, внедрение, эксплуатация. Современное планирование и управление ресурсами предприятия / Д. О'Лири. М.: Вершина, 2003. 278 с.
12. Шеер А.-В. Моделирование бизнес-процессов / А.-В. Шеер. М.: Весть-Метатехнология, 2000. 175 с.
13. Шеер А.-В. Бизнес-процессы. Основные понятия. Теория. Методы / А.-В. Шеер. М.: Весть-Метатехнология, 1999. 207 с.

УДК 72.01:005.2(075.8)

ББК 85.11я73

Б76



Издание осуществлено в рамках  
Инновационной образовательной программы ГУ ВШЭ  
«Формирование системы аналитических компетенций  
для инноваций в бизнесе и государственном управлении»

**Рецензенты:**

заведующий кафедрой прикладной математики

НГТУ имени Р.Е. Алексеева,

доктор физико-математических наук, профессор *С.И. Митяков*;

доктор физико-математических наук, профессор *Ю.М. Максимов*

ISBN 978-5-7598-0488-8

© Бабкин Э.А., 2007

© Козырев О.Р., 2007

© Оформление. Издательство «ТЕИС», 2007

**Бабкин, Э. А., Козырев, О. Р.** Архитектура и технология использования современных ERP-систем [Текст] : учеб. пособие / Э. А. Бабкин, О. Р. Козырев ; Гос. ун-т — Высшая школа экономики. — М. : ТЕИС, 2007. — 414 [2] с. — 1000 экз. — ISBN 978-5-7598-0488-8 (в пер.).

Учебное пособие предназначено для изучения курса «Архитектура и технология использования корпоративных информационных систем» по программе обучения студентов магистров направления «Бизнес-информатика». Рассматриваются вопросы практической разработки корпоративных информационных систем на примере широко известной системы с открытым кодом OFBIZ. Теоретический материал иллюстрируется примерами, даны задания для самостоятельных проектов студентов.

УДК 72.01:005.2(075.8)

ББК 85.11я73

OCR by Palek

*Учебное издание*

**Бабкин Эдуард Александрович  
Козырев Олег Рамазанович**

**АРХИТЕКТУРА  
И ТЕХНОЛОГИЯ ИСПОЛЬЗОВАНИЯ  
СОВРЕМЕННЫХ ERP-СИСТЕМ**

Редактор *Е.Л. Наумова*  
Художественный редактор *Н.Ф. Бердавцева*  
Компьютерная верстка: *А.А. Григорьев*  
Корректор *Е.И. Максакова*

Издано при содействии ООО «МАКС Пресс»

Подписано в печать 27.11.07. Формат 60×88/16  
П. л. 26,0. Уч.-изд. л. 22,26  
Тираж 1000 экз. Заказ № 1346

Издательство «ТЕИС»  
115407, Москва, Судостроительная ул., 59

Отпечатано в ППП «Типография «Наука»  
121099, Москва, Шубинский пер., 6