ИННОВАЦИОННАЯ ОБРАЗОВАТЕЛЬНАЯ ПРОГРАММА

Проект 2: индивидуальная траектория обучения и качество образования
Цель: ориентированное на требования рынка образовательных услуг улучшение качества подготовки и переподготовки специалистов

Федеральное агентство по образованию

Государственное образовательное учреждение
высшего профессионального образования

Владимирский государственный университет

Д.В. АЛЕКСАНДРОВ, Н.Н. ЖЕБРУН,
И.В. ГРАЧЕВ

РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ, ОСНОВАННЫЕ НА ЗНАНИЯХ

Практикум

Владимир 2008

УДК 681.518 (076)
ББК 32.988-5 я7
Р24

Рецензенты:
Доктор технических наук, профессор
зав. кафедрой информационных систем и информационного менеджмента
Владимирского государственного университета
А.В. Костров

Кандидат технических наук, доцент
зав. кафедрой информационного сервиса
Костромского государственного университета им. Н.А. Некрасова
А.Р. Денисов

Печатается по решению редакционного совета
Владимирского государственного университета

Александров, Д. В. Распределенные информационные системы,
Р24 основанные на знаниях : практикум / Д. В. Александров, Н. Н. Жебрун, И. В. Грачев ; Владим. гос. ун-т. – Владимир : Изд-во Владим. гос. ун-та, 2008. – 89 с. – ISBN 978-5-89368-874-0.

Рассматриваются практические аспекты построения и использования распределенных информационных систем, основанных на знаниях, современные системы управления базами данных (*Oracle*), мультиагентные системы (*JADE*), технологии веб-сервисов и управления бизнес-процессами предприятия (*Active BPEL*), системы групповой работы (*Microsoft SharePoint*), вопросы построения онтологий с помощью языка *OWL*.

Предназначен для студентов дневной и заочной форм обучения специальности 230201 – Информационные системы и технологии при выполнении лабораторных занятий по курсу «Представление знаний в информационных системах», магистрантов направления 230200 – «Информационные системы» при выполнении курсовой работы по дисциплине «Распределенные информационные системы», практических занятий по дисциплинам «Администрирование в распределенных автоматизированных системах» и «Информационный менеджмент», а также для аспирантов специальности 051301 – Системный анализ, управление и обработка информации (промышленность) при выполнении практических и лабораторных занятий по дисциплинам «Информационные технологии в науке и образовании» и «Распределенные информационные системы». Рекомендуется преподавателям при проведении занятий со студентами, магистрантами и аспирантами.

Ил. 27. Библиогр.: 7 назв.

УДК 681.518 (076)
ББК 32.988-5 я7

ISBN 978-5-89368-874-0

ПРЕДИСЛОВИЕ

В настоящее время практически все большие программные системы являются распределенными. Распределенная система – система, в которой обработка информации сосредоточена не на одной вычислительной машине, а распределена между несколькими компьютерами. При проектировании распределенных систем, которое имеет много общего с проектированием программного обеспечения в общем, все же следует учитывать некоторые специфические особенности.

Существует шесть основных характеристик распределенных систем.

1. Совместное использование ресурсов. Распределенные системы допускают совместное использование как аппаратных (жестких дисков, принтеров), так и программных (файлов, компиляторов) ресурсов.
2. Открытость. Это возможность расширения системы путем добавления новых ресурсов.
3. Параллельность. В распределенных системах несколько процессов могут одновременно выполняться на разных компьютерах в сети. Эти процессы могут взаимодействовать во время их выполнения.
4. Масштабируемость. Под масштабируемостью понимается возможность добавления новых свойств и методов.
5. Отказоустойчивость. Наличие нескольких компьютеров позволяет дублирование информации и устойчивость к некоторым аппаратным и программным ошибкам. Распределенные системы в случае ошибки могут поддерживать частичную функциональность. Полный сбой в работе системы происходит только при сетевых ошибках.
6. Прозрачность. Пользователям предоставляется полный доступ к ресурсам в системе, в то же время от них скрыта информация о распределении ресурсов по системе.

Распределенные системы обладают и рядом недостатков.

1. Сложность. Намного труднее понять и оценить свойства распределенных систем в целом, их сложнее проектировать, тестировать и обслуживать. Также производительность системы зависит от скорости работы сети, а не отдельных процессоров. Перераспределение ресурсов может существенно изменить скорость работы системы.
2. Безопасность. Обычно доступ к системе можно получить с нескольких разных машин, сообщения в сети могут просматриваться и перехватываться. Поэтому в распределенной системе намного труднее поддерживать безопасность.
3. Управляемость. Система может состоять из разнотипных компьютеров, на которых могут быть установлены различные версии операционных систем. Ошибки на одной машине могут распространяться непредсказуемым образом на другие машины.
4. Непредсказуемость. Реакция распределенных систем на некоторые события непредсказуема и зависит от полной загрузки системы, ее организации и сетевой нагрузки. Эти параметры постоянно изменяются, поэтому время ответа на запрос в каждом случае различное.

Из указанных недостатков следует, что при проектировании распределенных систем возникает ряд проблем, которые надо учитывать разработчикам.

1. *Идентификация ресурсов*. Ресурсы в распределенных системах располагаются на разных компьютерах, поэтому систему имен ресурсов следует продумать так, чтобы пользователи могли без труда открывать необходимые им ресурсы и ссылаться на них. Примером может служить система *URL* (унифицированный указатель ресурсов), которая определяет имена *Web*-страниц.
2. *Коммуникация*. Универсальная работоспособность Интернета и эффективная реализация протоколов *TCP*/*IP* для большинства распределенных систем служат примером наиболее эффективного способа организации взаимодействия между компьютерами. Однако в некоторых случаях, когда требуется особая производительность или надежность, возможно использование специализированных средств.

3. *Качество системного сервиса*. Этот параметр отражает производительность, работоспособность и надежность. На качество сервиса влияет ряд факторов: распределение процессов, ресурсов, аппаратные средства и возможности адаптации системы.
4. *Архитектура программного обеспечения (ПО)*. Архитектура ПО описывает распределение системных функций по компонентам системы, а также распределение этих компонентов по процессорам. Если необходимо поддерживать высокое качество системного сервиса, выбор правильной архитектуры является решающим фактором.

Задача разработчиков распределенных систем – спроектировать программное и аппаратное обеспечение так, чтобы предоставить все необходимые характеристики распределенной системы. А для этого требуется знать преимущества и недостатки различных архитектур распределенных систем. Выделяются три типа архитектур распределенных систем.

1. *Архитектура клиент/сервер*. В этой модели систему можно представить как набор сервисов, предоставляемых серверами клиентам. В таких системах серверы и клиенты значительно отличаются друг от друга.
2. *Трехзвенная архитектура*. В этой модели сервер предоставляет клиентам сервисы не напрямую, а посредством сервера бизнес-логики.
3. *Архитектура распределенных объектов*. В этом случае между серверами и клиентами нет различий и систему можно представить как набор взаимодействующих объектов, местоположение которых не имеет особого значения. Между поставщиком сервисов и их пользователями не существует различий.

Издание посвящено практическим аспектам построения (занятия № 2, 3, 4), администрирования (занятия № 1, 6) и использования распределенных информационных систем, основанных на знаниях (занятия № 5, 7). Практикум собрал опыт авторских работ, которые прошли апробацию в реальном учебном процессе по нескольким дисциплинам.

СПИСОК ОСНОВНЫХ СОКРАЩЕНИЙ

БД – база данных

ОС – операционная система

ПО – программное обеспечение

СУБД – система управления базами данных

ACL – Agent Communication Language

BPEL – Business Process Execution Language

DF – Directory Facilitator

FIPA – Foundation for Intelligent Physical Agents

JADE – Java Agent Development Framework

JDK – Java Development Kit

OWL – Ontology Web Language

URL – Unified Resource Locator

WSDL – Web Services Description Language

WSPS – Windows Share Point Services

Практическое занятие №1
Основы администрирования СУБД *Oracle*

Цель работы: ознакомиться с основами администрирования СУБД *Oracle* на примере СУБД *Oracle* версии *10g*. Выполнить процесс установки серверной и клиентских частей СУБД *Oracle*. Изучить возможности основных инструментов администрирования СУБД. Создать учебную БД с помощью клиентских инструментов и выполнить удаленный запрос к таблицам БД.

Теоретические сведения

9 сентября 2003 года корпорация *Oracle* представила новую версию своей СУБД – *Oracle Database* 10*g*. Кроме того, *Oracle* представила такие новые продукты, как *Oracle Application Server* 10*g* и *Oracle Enterprise Manager* 10*g*. Все вместе они составляют программный комплекс *Oracle Grid Computing* – интегрированное программное обеспечение для инфраструктуры распределенных вычислений.

Oracle Grid Computing позволяет использовать корпоративные программные пакеты, такие как *Oracle E-Business Suite*, и выполнять другие приложения сторонних поставщиков программного обеспечения, например *PeopleSoft* и *SAP*.

Новое решение позволит клиентам создавать крупномасштабные вычислительные инфраструктуры из недорогих стандартных компонентов, таких как кластеры серверов-лезвий и смонтированные в стойку системы хранения данных. Для решения этой задачи в *Oracle Database* 10*g* включены усовершенствованные возможности автоматизированного управления и новая консоль *Database Control* с *Web*-интерфейсом. Основанная на *Oracle Enterprise Manager* 10*g* консоль *Database Control* представляет собой диагностический монитор, отображающий в графическом виде текущее состояние функционирующей базы данных.

Кроме того, возможность *Oracle Database* 10*g* осуществлять автоматическое управление избавляет от необходимости выполнять сложные повторяющиеся задачи, такие как диагностика производительности, настройка приложений и управление распределением памяти. Новое программное обеспечение включает механизм самодиагностики, который автоматически выявляет проблемы, связанные с производительностью или функциональностью системы, и предоставляет рекомендации по устранению неполадок.

В основе решения *Oracle Grid Computing* лежит технология кластеризации баз данных *Oracle Real Application Clusters*. Сегодня это единственное программное обеспечение для кластерных баз данных, которое позволяет использовать пакеты приложений без внесения в них каких-либо изменений при пополнении кластера новыми серверами или устройствами хранения данных.

С появлением *Oracle Real Application Clusters* 10*g* развертывание и управление кластерами баз данных как сетью распределенных вычислений предприятия должно стать гораздо проще за счет использования интегрированного кластерного программного обеспечения (*integrated clusterware*). Это программное обеспечение представляет собой набор общих сервисов по обслуживанию кластеров, встроенных в *Oracle Database* 10*g* с целью облегчения создания кластеров и управления ими. До последнего времени использование технологии *Oracle Real Application Clusters* на различных платформах *ОС UNIX*, *Linux*, *Windows* требовало глубоких технических знаний. *Oracle Real Application Clusters* 10*g* включает все кластерное программное обеспечение, необходимое для легкой установки, конфигурации и эффективной работы кластеров на всех поддерживаемых платформах.

Oracle Real Application Clusters 10*g* также включает новое ПО для управления рабочей нагрузкой кластеров, предназначенное для перераспределения вычислительных возможностей компонентов кластерной базы данных.

Oracle Database 10*g* включает модуль автоматического управления хранилищем данных (*Automatic Storage Management*, *ASM*) – новое ПО, предназначенное для значительного упрощения конфигурирования системы хранения данных и управления базами данных.

СУБД *Oracle* всегда отличалась большим выбором функциональных возможностей и средств, состав и количество которых увеличиваются в каждом новом выпуске СУБД. Это помогает пользователям сделать систе-

му базы данных более надежной, мощной и эффективной, но рост функциональных возможностей СУБД также оказывает непосредственное влияние и на методы администрирования СУБД. Для того чтобы воспользоваться преимуществами новых функциональных возможностей, администраторы должны пересматривать используемые методы администрирования и приводить их в соответствие с каждой новой версией программного обеспечения.

Методы администрирования можно разделить на четыре категории:

1. Конфигурация систем и баз данных

- Конфигурация внешней памяти;
- Размер блоков базы данных;
- Метод создания базы данных;
- Сопровождение пространства и объектов;

2. Сопровождение пространства и объектов

- Автоматическое управление пространством отката транзакций;
- Локально управляемые табличные пространства;
- Временные табличные пространства;
- Сопровождение сегментов;

3. Оптимизация производительности

Оптимизация осуществляется различными способами в зависимости от причин низкой производительности. Такими причинами могут быть:

- Плохое управление соединениями;
- Плохое совместное использование курсоров;
- Неправильное конфигурирование ввода-вывода;
- Проблемы конфигурирования журнала базы данных;
- Нехватка списков свободных блоков, групп списков свободных блоков и сегментов отката;

- Длительные полные просмотры таблиц;
- Дисковые сортировки;
- Большие объемы рекурсивных операций *SQL*;
- Ошибки схем и оптимизатор;
- Использование нестандартных параметров инициализации;

4. Резервирование и восстановление

Умение эффективно использовать методы администрирования позволяет повысить производительность работы приложений с сервером БД.

Задание

1. Установить серверную часть СУБД *Oracle* «в расширенном» режиме.
2. Ознакомиться с основными инструментами управления СУБД *Oracle*.
3. Установить клиентскую часть СУБД *Oracle*.
4. С помощью клиентских инструментов создать новую схему и выполнить тестирование работоспособности СУБД путем выполнения запроса к созданной БД.

Предварительные требования

Для выполнения работы требуется дистрибутив СУБД *Oracle* 10*g*.

Порядок выполнения работы

1. Установить серверную часть СУБД *Oracle 10g*.
   1.1. Начать установку сервера *Oracle*, запустив с дистрибутива программу установки *setup.exe*.
   1.2. После запуска *Oracle Universal Installer* на первом шаге выбрать *Advanced Installation*.
   1.3. На втором шаге указать пути размещения файлов. В разделе *Source* указать *XML*-файл, содержащий информацию о устанавливаемых продуктах (*<DISTRIB_PATH>:\stage\products.xml*). В разделе *Destination* указать имя инсталляции и путь установки продуктов *Oracle*.

1.4. На третьем шаге после загрузки информации о продуктах выбрать тип инсталляции как *Enterprise Edition*.

1.5. На пятом шаге требуется указать инсталлятору, нужно ли создавать начальную базу (инстанцию), а также назначение БД, которое требуется для оптимизации работы СУБД с БД. Здесь можно указать *General Purpose* (БД для основных целей), *Transaction Processing* (БД с оптимизацией выполнения больших транзакций), *Data Warehouse* (БД, оптимизированная для длительного хранения данных), *Advanced* (позволяет администратору вручную выбрать конфигурацию БД). Выбрать *Advanced*.

1.6. На следующей экранной форме показаны параметры установки, языки установки, а также устанавливаемые продукты. Нажать кнопку *Install* для запуска процесса инсталляции сервера *Oracle*.

1.7. После процесса установки инсталлятор запускает *Database Configuration Assistant* – инструмент для создания БД *Oracle*.

2. Установленная серверная часть СУБД *Oracle* включает в себя следующие инструменты:

- инструменты разработки приложений (*Application Development*). Это документация по *Oracle* интерфейсам и объектам для разработчиков, а также расширенный *SQL* редактор *SQL Plus*;
- инструменты конфигурирования БД (*Configuration tools*). Сюда входят инструмент управления объектами *Oracle* (*Administration Assistant*), инструмент управления базами (*Database Configuration Assistant*), инструмент обновления баз прежних версий до версии 10*g* (*Database Upgrade Assistant*), а также инструменты создания локалей (*Locale Builder*) и управления сетевых настроек (настроек слушающих портов и сетевых сервисов) – *Net Manager* и *Net Configuration Assistant*;
- инструменты управления интеграцией – *Oracle Directory Manager* и утилита создания сертификатов *Wallet Manager*.

В данной работе воспользуемся инструментом *Database Configuration Assistant* для создания базы данных:

2.1. На первом шаге помощника *Database Configuration Assistant* выбрать операцию создания базы данных *Create a Database*.

2.2. На втором шаге выбрать шаблон создания БД как *General Purpose* (см. выше). Посмотреть детали шаблона можно, нажав кнопку *Show Details*.

2.3. На третьем шаге ввести имя глобальной базы данных (инстанции), а также её уникальный номер *SID* (например *MyDatabase*).

2.4. На следующем шаге необходимо выбрать способ управления базы данных либо с помощью *Enterprise Manager*, либо с помощью средств управления *Grid*. Выбрать *Configure the Database with Enterprise Manager*.

2.5. Следующий шаг позволяет ввести пароли для основных учетных записей системы (*SYS, SYSTEM, DBSNMP, SYSMAN*).

2.6. На следующем шаге в качестве механизма хранения базы данных указать *File System*.

2.7. Далее нужно указать пути к файлам базы данных либо использовать стандартные пути из шаблона. Здесь выбрать *Use Database File Locations from Template*.

2.8. Следующий шаг позволяет редактировать настройки области восстановления БД. Отключить опцию *Specify Flash Recovery Area*.

2.8. Далее помощник спросит о необходимости создания тестовых схем и запуска пользовательских скриптов. Пропустить данный шаг.

2.9. На следующем шаге можно указать параметры инициализации БД: размер выделяемой памяти, количество соединений, параметры кодировки и режимы соединений. В данной работе оставим все настройки по умолчанию.

2.10. Следующий шаг позволяет отредактировать файлы хранения данных. Пропустить его и нажать кнопку «Готово» для создания базы данных.

Чтобы проверить факт установки базы данных запустите *Web Enterprise Manager* по адресу *http://<HOST>:5500/em*.

3. Установить клиентскую часть СУБД *Oracle*.

3.1. Начать установку клиентской части СУБД *Oracle*, запустив с дистрибутива инсталлятор *setup.exe*.

3.2. На первом шаге *Oracle Universal Installer* указать пути размещения файлов. В разделе *Source* указать *XML*-файл, содержащий информацию об устанавливаемых продуктах (*<DISTRIB_PATH>:\stage\products.xml*). В разделе *Destination* указать имя инсталляции и путь установки продуктов *Oracle*.

3.3. На следующем шаге после загрузки информации о продуктах выбрать тип инсталляции как *Administrator*.

3.4. На следующей экранной форме показаны параметры установки, языки установки, а также устанавливаемые продукты. Нажать кнопку *Install* для запуска процесса инсталляции клиента *Oracle*.

3.5. После процесса установки инсталлятор запускает *Oracle Net Configuration Assistant* для установки сетевых сервисов.

4. Главный клиентский инструмент управления СУБД *Oracle – Enterprise Manager Console*, который позволяет производить основные действия над объектами базы данных *Oracle*. В работе необходимо ознакомиться с данным инструментом, для этого:

4.1. Запустить *Enterprise Manager Console*. При первом запуске необходимо добавить БД в дерево объектов менеджера. Для этого указать имя сервера (*hostname*), номер порта (обычно 1521) и уникальный идентификатор *Oracle SID* (например *MyDatabase*). В случае верно введенных данных БД добавится в дерево объектов.

4.2. Подсоединиться к БД, указав имя (*username*) как, например *system*, и пароль, указанный при установке базы данных.

4.3. Ознакомиться с узлом *Instance*, с помощью которого можно редактировать параметры базы данных (инстанции).

4.4. Ознакомиться с узлом *Schema*. С помощью этого узла можно управлять существующими схемами (базами данных в обычном понимании), добавлять, удалять таблицы, хранимые процедуры, триггеры и виды.

4.5. Ознакомиться с узлом *Security*. Здесь можно настраивать параметры безопасности БД, добавлять/редактировать/удалять пользователей, роли и профили.

Добавим нового пользователя в БД. Для этого в контекстном меню раздела *Users* выбрать *Create*. В форме создания пользователя указать следующее:

- имя (*name*) пользователя (например *Jack*) и пароль;
- в разделе *Role* добавить роль *DBA* (рис.1).

Создать пользователя, нажав кнопку *Create*.

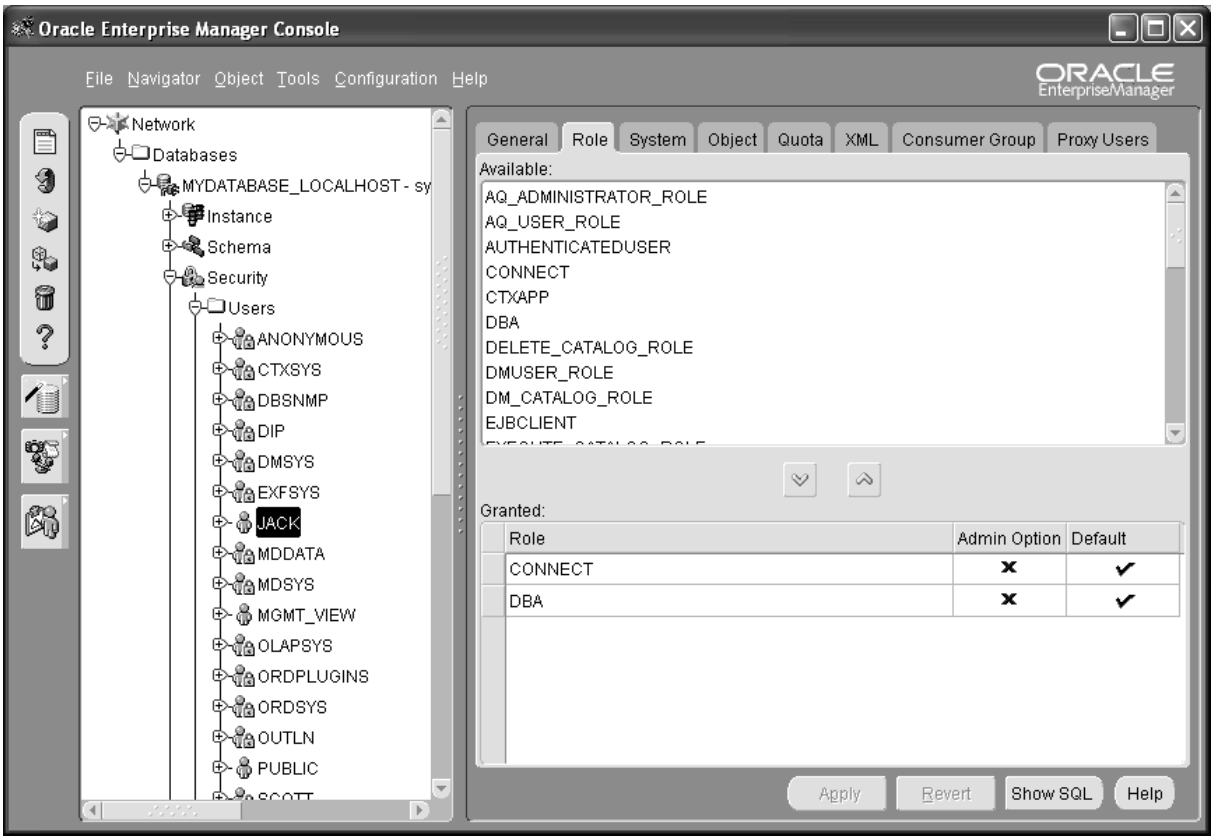

Рис.1. Форма управления ролями пользователя

4.5. Ознакомиться с узлом *Storage*. Этот узел позволяет настроить параметры физического хранения файлов базы данных.

4.6. Ознакомиться с узлом *Distributed*. С помощью этого узла можно настроить репликацию и связать распределенные БД.

4.7. Ознакомиться с узлом *Warehouse*. Этот узел позволяет управлять базой данных в качестве хранилища данных (этот узел особо важен для работы с БД *OLAP*-приложений).

4.8. Ознакомиться с узлом *Workspace*. Этот узел позволяет создавать рабочие области – виртуальные окружения, которые позволяют одному или нескольким пользователям разделять возможность изменения данных в базе.

4.9. Ознакомиться с узлом *XML Database*, который позволяет настраивать *XML*-базы данных.

4.10. Создать таблицу в схеме, которая была автоматически создана при добавлении пользователя. Для этого запустить *SQL Scratchpad* из меню *Enterprise Manager* и зайти под созданным пользователем. Создать таблицу с помощью *SQL* команды, например:

```
create table MyTable(ID int not null, Name varchar(50))
```

4.11. Создать тестовые записи в таблице и выполнить выборку записей с помощью утилиты *SQL Scratchpad*.

5. Оформить отчет в соответствии с требованиями.

Вопросы для самоконтроля

1. Как создать новую схему в существующей базе данных *Oracle*?
2. Какие средства предоставляет клиентская часть СУБД *Oracle*?
3. Какие средства предоставляет серверная часть СУБД *Oracle*?
4. Какие возможности предлагает СУБД *Oracle* с точки зрения использования базы как хранилища данных?

Практическое занятие № 2
Создание бизнес-процесса в *BPEL*

Цель работы: получить навыки реализации прототипа распределенного бизнес-процесса для выбранной предметной области на основе технологии *BPEL* и веб-сервисов.

Задание

1. Выбрать предметную область и продумать содержание бизнес-процесса и веб-сервиса, который будет участвовать в процессе.
2. Нарисовать диаграмму бизнес-процесса.
3. Создать *WSDL*-описание бизнес-процесса и веб-сервиса, который им используется.

Реализовать *BPEL*-процесс и смоделировать его работу в среде *ActiveBPEL Designer*.

Предварительные требования

Для выполнения работы на компьютере должно быть установлено следующее ПО:

- *JDK* версии 1.5 или выше.
- *Eclipse IDE* версии 3.2.1 или выше с установленными расширениями *J2EE Standard Tools* 1.5.2 или выше и *Web Standard Tools* 1.5.2 или выше.
- *ActiveBPEL Designer* версии 2.0.

Для поиска публичного веб-сервиса потребуется доступ в Интернет.

При реализации бизнес-процесса потребуются знания технологии веб-сервисов и в частности *WSDL*.

Теоретические сведения

По мнению ведущих *IT*-компаний и аналитиков (наиболее известные из них – *IBM*, *Microsoft*, *Sun Microsystems*, *BEA*, *SAP*, *Oracle*, *Gartner Group*, *Stencil Group*, *International Data Corp*.) важными и перспективными на-

правлениями в развитии информационных систем и ПО являются архитектуры, ориентированные на сервисы (*Service Oriented Architecture – SOA*). При этом основной акцент делается на *SOA*, которая ориентирована на Интернет, т.е. на архитектуру веб-сервисов (*Web Services Architecture – WSA*).

Архитектура, ориентированная на сервисы, имеет следующие характерные особенности:

1. Архитектура является распределенной. Функциональные модули приложения (системы) могут быть распределены по множеству вычислительных систем и способны к взаимодействию с использованием локальных или глобальных сетей.

2. Интерфейс функциональных модулей таков, что использование модулей не зависит от технологии или платформы, в рамках которой они реализованы.

3. Возможен динамический поиск и подключение нужных функциональных модулей.

4. Архитектура базируется на общепринятых отраслевых стандартах.

В состав *SOA* входят три компонента: поставщик сервиса, потребитель сервиса и реестр сервисов.

Поставщик сервиса регистрирует его в некотором реестре, где его обнаруживает потребитель, после чего между ними устанавливается связь в соответствии с контрактом и осуществляется требуемое действие.

Для реализации *SOA* необходимы три типа соглашения:

1. Транспортное соглашение: о форматах и протоколах взаимодействия.

2. Соглашение об описании функциональности сервиса в виде, пригодном для автоматической генерации кода, который определяет процесс взаимодействия между клиентом и поставщиком сервиса.

3. Соглашение о способе обнаружения сервиса.

Архитектура веб-сервисов является одной из реализаций *SOA*. Кроме нее существуют следующие походы к ее реализации: *Java RMI* (от *Sun Microsystems*), *CORBA* (от консорциума *OMG*), *DCOM* (от *Microsoft*), *DCE* (предложенный ассоциацией *Open Group*).

Концепция веб-сервисов возникла в конце 90-х годов XX века. Однако к настоящему моменту эта концепция успела устояться и архитектура, которую она предлагает, стала отраслевым стандартом в сфере *IT*.

Стандартизацией архитектуры веб-сервисов занимаются рабочие группы комитета *W3C*. Они предлагают следующее определение понятия «веб-сервис»: «веб-сервис – это реализуемая программными средствами система для поддержки межмашинного взаимодействия через сеть. Интерфейс сервиса описывается на языке, читаемом машиной, например *WSDL*. Другие системы взаимодействуют с веб-сервисом способом, указанным в его описании, используя сообщения в стандарте *SOAP*, передаваемые с использованием *HTTP* и *XML* и в сочетании с другими стандартами, относящимися к *Web*».

Физически *Web*-сервис представляет собой фрагмент программного обеспечения, называемый «агентом». Агент способен передавать и принимать сообщения, он реализует функциональность сервиса. Существует различие между агентом и сервисом – один и тот же сервис может быть обеспечен разными агентами.

Механизм обмена сообщениями определяется в описании сервисов (*Web Services Description*), которое представляет собой спецификацию интерфейса сервиса и охватывает форматы сообщений, типы данных, транспортные протоколы, способы сериализации, используемые при обмене между агентами заказчика и поставщика услуг. Кроме того, описание сервиса содержит указание на одну или несколько точек в сети (*endpoint*), откуда доступен поставщик.

Технология *Universal Description*, *Discovery and Integration* (*UDDI*) предполагает ведение реестра веб-сервисов. Подключившись к этому реестру, потребитель сможет найти веб-сервисы, которые удовлетворяют его потребностям. Технология *UDDI* дает возможность поиска и публикации нужного сервиса как человеком, так и программой-клиентом.

Business Process Execution Language for Web Services (*BPEL for Web Services* или *BPEL4WS*) представляет собой язык описания бизнес-процессов, ориентированный на их автоматизированное выполнение путем взаимодействия веб-сервисов различных участников процесса.

В данной спецификации выделяются две разновидности бизнес-процессов. Выполняемые бизнес-процессы моделируют поведение участника бизнес-взаимодействия. В свою очередь бизнес-протоколы используют описания процессов, которые специфицируют взаимно видимые обмен сообщениями, поведение каждого вовлеченного в протокол участника без открывания их внутреннего поведения. Описания процессов для бизнес-

протоколов называются абстрактными процессами. *BPEL* предназначен для моделирования как выполняемых, так и абстрактных процессов.

BPEL представляет собой язык формальной спецификации бизнес-процессов и протоколов бизнес-взаимодействий, базирующийся на *XML*. Таким образом, он расширяет модель взаимодействия веб-сервисов и вводит в нее поддержку бизнес-транзакций. Также *BPEL* определяет интероперабельную интеграционную модель, которая призвана обеспечивать интеграцию автоматизированных процессов как внутри предприятия, так и на уровне взаимодействия партнеров по бизнесу.

В *BPEL* выделен ряд базовых концепций, первая из которых – концепция «партнеров». Партнерами являются внешние веб-сервисы, включенные организацией в свои бизнес-процессы. Информация, которой обмениваются партнеры, описывается в «контейнерах». «Деятельности» – это действия или задачи, выполняемые в рамках бизнес-процесса. Наконец, «обработчики ошибок» представляют собой особые деятельности, в их задачи входит обработка ошибок, происходящих во время выполнения других действий.

Порядок выполнения работы

Выбор предметной области

Выберите предметную область и постройте на черновике диаграмму бизнес-процесса в свободной форме. Определите на диаграмме входные и выходные данные процесса и отдельных блоков и условия перехода.

К бизнес-процессу предъявляются следующие требования:

- Наличие в процессе одного блока, вызывающего веб-сервис, который затем будет реализован самостоятельно, и блока, вызывающего сторонний веб-сервис;
- Наличие хотя бы одного ветвления или цикла.

Примеры предметных областей:

- Бизнес-процесс выдачи кредита включает вызов сервиса оценки риска (*Assessor*) в случае крупной суммы кредита и вызов сервиса, подтверждающего выдачу кредита (*Approver*). Этот пример включен в комплект поставки *ActiveBPEL Designer*;

- Бизнес-процесс заказа туристической путевки может включать вызов сервиса, показывающего текущую погоду на выбранном курорте, и вызов сервиса бронирования комнаты в отеле;
- Бизнес-процесс покупки товара в Интернет-магазине может включать проверку платежеспособности кредитной карты, указанной покупателем, и проверку наличия товара на складе.

Исходя из требования использовать в бизнес-процессе сторонний веб-сервис, полезно сначала найти подходящий веб-сервис, а уже затем выбирать предметную область под него.

Поиск стороннего веб-сервиса

Рекомендуется с целью облегчения отладки бизнес-процесса в среде *ActiveBPEL Designer* использовать только веб-сервисы в стиле *RPC/Encoded*. Далее будет рассматриваться работа только с такими сервисами.

Искать веб-сервисы можно в поисковых системах Интернета. Существуют также специальные сайты, на которых есть списки публичных веб-сервисов, например *http://xmethods.net/*. При выборе веб-сервиса обращайте внимание на стиль (*RPC/Encoded*) и на условия использования веб-сервиса, так как там могут быть ограничения на количество вызовов в сутки. Для проверки работоспособности веб-сервиса удобно использовать *Web Services Explorer*, встроенный в среду *Eclipse* (рис. 2). Для этого:

1. Выберите пункт меню *Run → Launch the Web Services Explorer*.
2. Нажмите кнопку *WSDL Page*.
3. В области *Navigator* нажмите на *WSDL Main*.
4. Введите в поле *WSDL URL* ссылку на *WSDL*-описание сервиса и появится возможность вызвать любую операцию веб-сервиса с заданными параметрами. В некоторых *WSDL*-описаниях встречается встроенная документация для каждой операции.

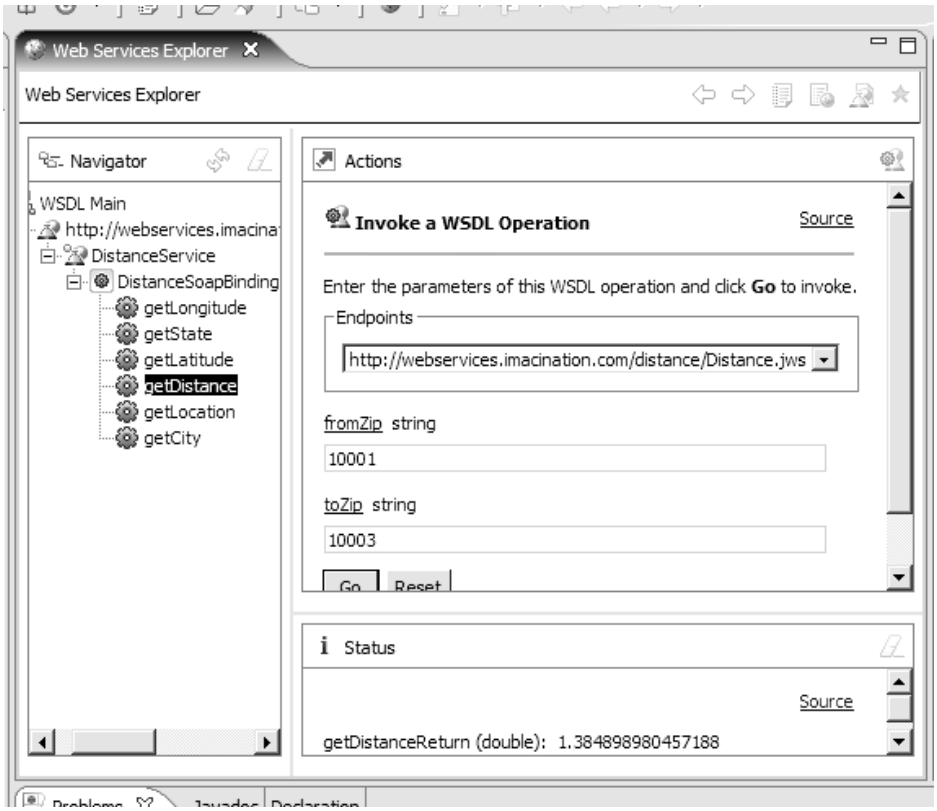

Рис. 2. Интерфейс Web Services Explorer

Создание WSDL-описания

Взяв за основу результаты выполнения предудыщих пунктов, создайте два отдельных *WSDL*-описания: для вашего веб-сервиса и для *BPEL*-процесса. Если Вы хорошо знаете *WSDL*, то сделать это можно в любом текстовом редакторе. Однако более удобным представляется использование средств, предоставляемых средой *Eclipse*.

Запустите среду *Eclipse*.

1. Создайте новый проект (*File → New → Project*) типа *Dynamic Web Project*. Укажите имя проекта, соответствующее выбранной предметной области. Остальные настройки можно оставить по умолчанию.
2. Создайте новое *WSDL*-описание (*File → New → Other → Web Services → WSDL*). Задайте имя *WSDL*-файла, *URI* целевого пространства имен *XML* (*Target Namespace*), префикс пространства имен. Выберите в пункте *SOAP Binding Options* пункт *rpc encoded*.

Появится предупреждение о том, что *rpc encoded* не поддерживается спецификациями *WS-I*, которое можно проигнорировать. В результате будет сгенерирован шаблон *WSDL*-описания.

3. Поправьте сгенерированный средой шаблон *WSDL*-описания в графическом режиме или в текстовом. При этом адрес привязки для сервиса *BPEL*-процесса должен быть *вида http://localhost:8080/active-bpel/services/<имя-BPEL-сервиса>*, а для вашего сервиса вида *http://localhost:8080/<имя-проекта>/services/<имя-порта-сервиса>* (рис. 3). Также надо изменить набор аргументов операции, их типы и имена в соответствии с предметной областью. Проследите, чтобы в *WSDL*-описании не было сущностей с одинаковыми именами, так как это может впоследствии затруднить реализацию веб-сервиса.

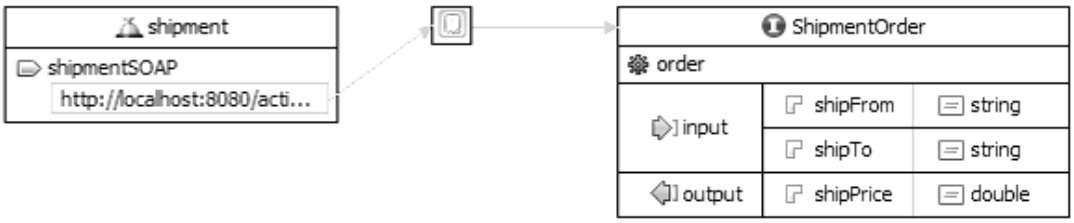

Рис. 3. Графическое представление получившегося WSDL-описания

Реализация BPEL-процесса

Запустите среду *ActiveBPEL Designer*.

1. Создайте новый проект (*File → New → Project*) типа *Simple Project*. Задайте имя проекта, соответствующее выбранной предметной области.
2. Сохраните в папку проекта *WSDL*-описание стороннего веб-сервиса для ускорения работы с ним.
3. Добавьте *WSDL*-описание бизнес-процесса и всех используемых в нем веб-сервисов в среду. Для этого перейдите на вкладку *Web References*, нажмите на кнопку *Add a Web Reference*, укажите местонахождение *WSDL*-файла и все описания из него будут доступны для работы. Проделайте это действие для всех трех *WSDL*-описаний, используемых в работе.

4. Добавьте отладочные значения для переменных, которые будут использоваться в *BPEL*-процессе. Это понадобится в дальнейшем для моделирования работы процесса. На вкладке *Web References* выберите в меню пункт *View Messages*. В контекстном меню переменной выберите пункт *Add Sample* и укажите значение.
5. Создайте новый *BPEL*-процесс (*File → New → BPEL Process*). Задайте имя процесса, соответствующее выбранной предметной области.
6. Поместите в *BPEL*-процесс блоки *Receive* и *Reply*. Для этого перейдите на вкладку *Web References* и выберите в меню *View WSDL*. Найдите порт и операцию, соответствующую *BPEL*-процессу, и перетащите ее в поле редактирования процесса. Появится мастер, предлагающий создать новый *Partner Link Type*, задайте имя для него и *Role*. Далее выберите *Add To WSDL File* и укажите *WSDL*-файл *BPEL*-процесса. В окне *Activity Type* выберите *Receive-Reply*. Будет создано два блока, соответствующих входу и выходу бизнес-процесса.
7. Аналогично перетащите операции двух оставшихся веб-сервисов в поле редактирования бизнес-процесса, но теперь выберите в окне *Activity Type* тип *Invoke*.
8. Используя вкладку *Properties*, задайте имена для всех созданных блоков, изменив содержимое свойства *Activity Name*.
9. Соедините блоки линиями перехода. Для этого выделите два блока и в контекстном меню выберите пункт *Link Activities*.
10. Задайте условия для некоторых переходов. Для этого выделите переход и выберите пункт контекстного меню *Edit Transition*. Появится окно, в котором можно выбрать переменную из имеющихся и условие перехода.
11. Так как переменные различных блоков не согласованы между собой, необходимо добавить несколько блоков типа *Assign* для того, чтобы приравнять значение выходной переменной одного блока к входной переменной другого. Для этого перейдите на вкладку *Process Variables*, найдите там нужную переменную и выберите в контекстном меню *Open*. В открывшемся поле найдите часть, которую нужно скопировать, и выберите пункт контекстного меню *Copy → From* или

Copy → *To*, в появившемся окне выберите другую часть операции присваивания. В поле редактирования бизнес-процесса появится новый блок, который надо вставить в нужное место процесса и связать линиями перехода.

12. Выберите блок *Receive* и в его свойствах установите значение *Create Instance*, равное *Yes* (рис. 4).

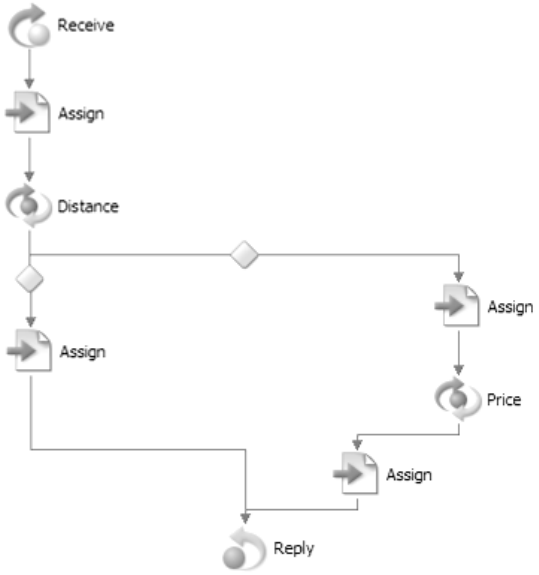

Рис. 4. Диаграмма BPEL-процесса

13. Сохраните проект. Если процесс построен правильно, то на вкладке *Problems* не должно быть никаких сообщений.

Моделирование BPEL-процесса

Моделирование *BPEL*-процесса в среде *ActiveBPEL Designer* можно запустить, нажав на кнопку панели инструментов *Start Simulation of the Process*. При этом среда перейдет в режим отладки. Используйте кнопку *Step Over* на вкладке *Debug* для пошагового выполнения процесса. Текущие значения переменных можно увидеть на вкладке *Process Variables*, выберите пункт контекстного меню *Open All*, чтобы открыть все используемые переменные для просмотра. На вкладке *Console* выводятся отладочные сообщения о ходе выполнения процесса. Чтобы убрать с диаграммы информацию о текущем выполнении процесса, нажмите кнопку, находящуюся рядом с кнопкой запуска моделирования.

Промоделируйте *BPEL*-процесс с разными значениями переменных, чтобы проверить различные пути выполнения.

Пример выполнения

Предметная область: заказ транспортных перевозок в пределах США. На входе – почтовый индекс места отправления и места прибытия. На выходе – цена за один контейнер.

Сервис *http://webservices.imacination.com/distance/Distance.jws?wsdl* определяет расстояние по двум почтовым индексам. На выходе – расстояние в милях или 0, если указаны некорректные данные.

Собственный сервис: определение стоимости. На входе – расстояние, на выходе – цена в долларах, вычисляемая как расстояние, помноженное на 50 (рис. 5).

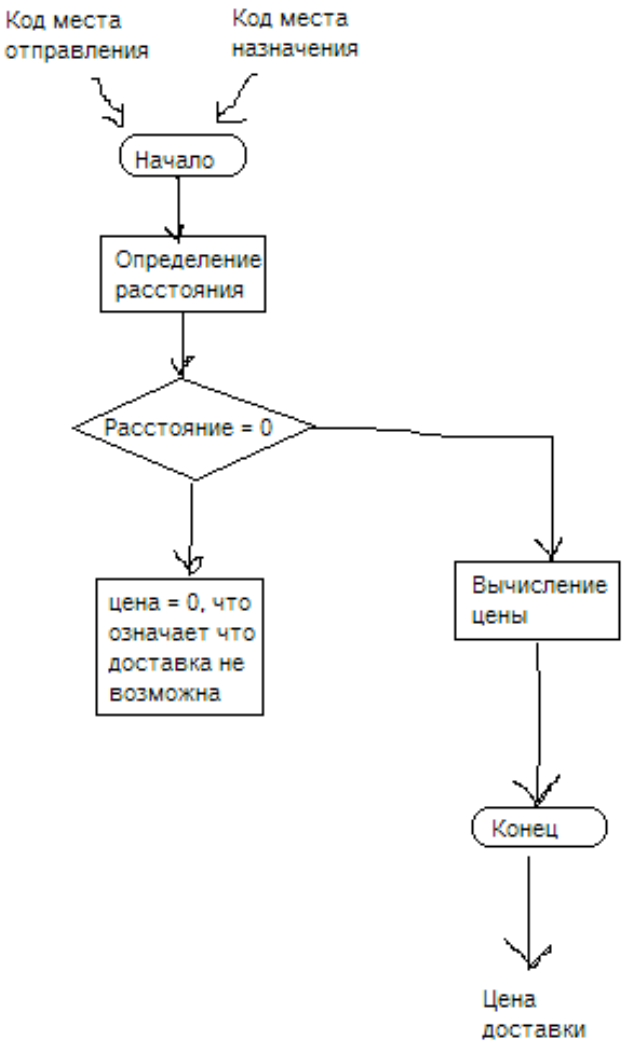

Рис. 5. Черновик диаграммы процесса

WSDL-описание для *BPEL*-процесса – см. рис. 3.
WSDL-описание для собственного сервиса:

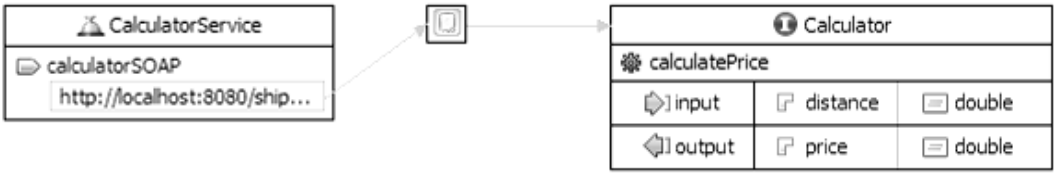

Диаграмма *BPEL*-процесса – см. рис. 4.
Процесс моделирования представлен на рис. 6.

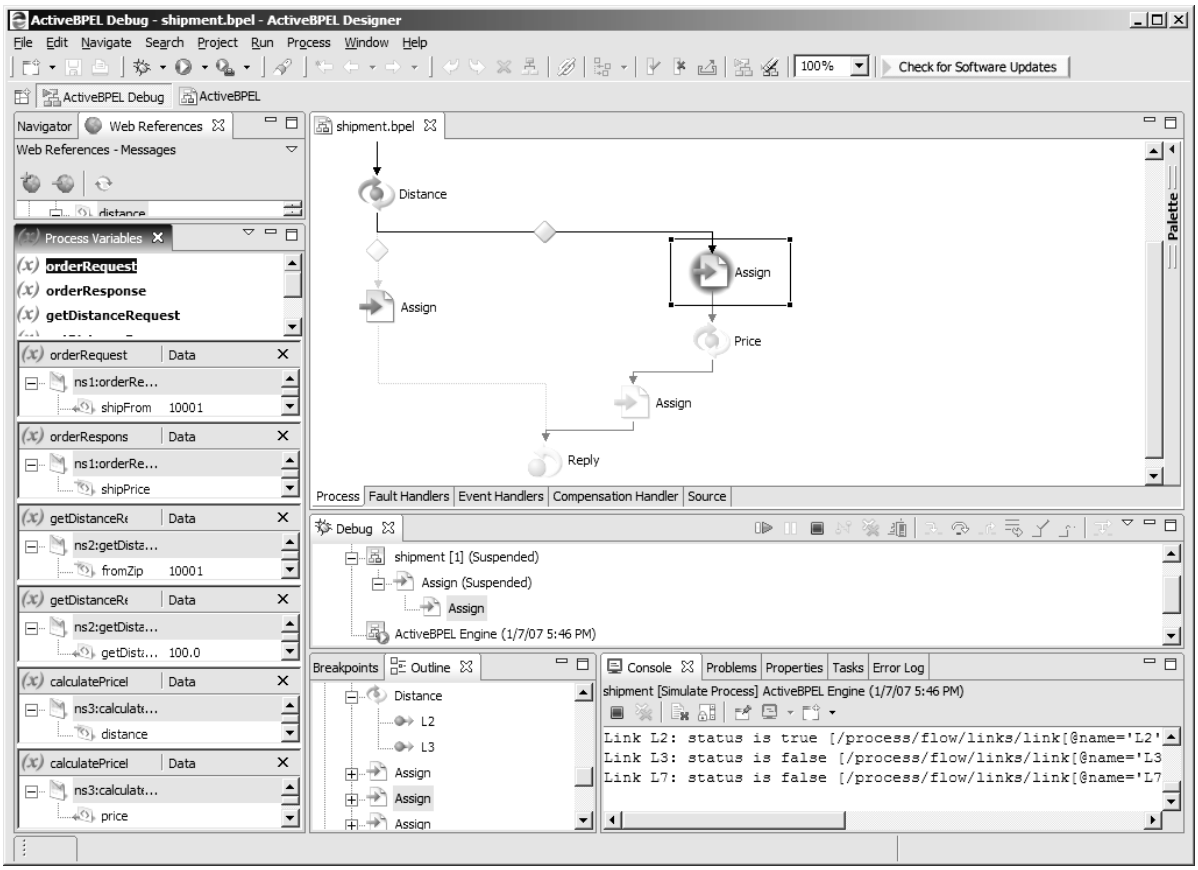

Рис. 6. Снимок экрана в процессе моделирования

Вопросы для самоконтроля

1. Чем *BPEL*-процесс отличается от веб-сервиса? В чем сходства?
2. Какие факторы нужно учитывать при принятии решения о том, как реализовать бизнес-процесс – на языке *BPEL* или в виде *Java*-веб-сервиса, вызывающего другие сервисы?
3. Какие сложности могут возникнуть при реализации на языке *BPEL* бизнес-процесса, включающего человеко-машинные взаимодействия? Предложите способы их преодоления.
4. Что нужно знать, чтобы воспользоваться публичным веб-сервисом?
5. Какую информацию можно найти в *WSDL*-описании веб-сервиса?
6. Перечислите основные типы блоков *BPEL*-процесса и их назначение.
7. Как избавиться от необходимости вставлять блоки *Assign* между вызовами веб-сервисов?

Практическое занятие № 3
Создание веб-сервиса

Цель работы: получить навыки создания и развертывания веб-сервиса с использованием среды *Eclipse*, сервера приложений *Apache Tomcat* и контейнера *Apache Axis*, а также развертывания и удаленной отладки *BPEL*-процесса в среде *ActiveBPEL Designer*.

Задание

1. Реализовать веб-сервис по имеющемуся *WSDL*-описанию.
2. Выполнить развертывание веб-сервиса на сервере приложений и испытать его работоспособность.
3. Выполнить развертывание имеющегося *BPEL*-процесса.
4. Проверить работоспособность *BPEL*-процесса и осуществить пошаговое выполнение процесса на сервере.

Предварительные требования

Для выполнения работы на компьютере должно быть установлено следующее ПО:

- *JDK* версии 1.5 или выше.
- *Eclipse IDE* версии 3.2.1 или выше с установленными расширениями *J2EE Standard Tools* 1.5.2 или выше и *Web Standard Tools* 1.5.2 или выше.
- *ActiveBPEL Designer* версии 2.0.

В ходе выполнения работы потребуется следующая информация об установленном ПО:

Папка установки сервера приложений, входящего в комплект *ActiveBPEL Designer*.

Переменная окружения *JAVA_HOME* должна указывать на папку, в которую установлен *JDK* (это требуется при запуске сервера приложений).

Для успешного запуска сервера приложений требуются файлы *xalan.jar* и *serializer.jar* из дистрибутива *Xalan-Java*. Их следует поместить в папку *common\lib* сервера приложений.

Для проверки работоспособности *BPEL*-процесса, вызывающего удаленный веб-сервис, потребуется доступ в Интернет.

При реализации бизнес-процесса потребуются знания технологии веб-сервисов и в частности *WSDL*, а также знание языка *Java*.

Порядок выполнения работы

Реализация веб-сервиса

Взяв за основу *WSDL*-описание, созданное в предыдущей лабораторной работе, сгенерируйте необходимые заглушки и напишите класс реализации веб-сервиса. Для этого воспользуйтесь мастером создания веб-сервисов среды *Eclipse*.

1. Запустите среду *Eclipse* и откройте проект, созданный в предыдущей лабораторной работе.
2. Создайте новый веб-сервис (*File → New → Other → Web Services → Web Service*). В поле *Web Service Type* выберите *Top down Java bean Web Service*. Укажите расположение *WSDL*-файла. Установите ползунок на уровень *Deploy Service*. Будут сгенерированы все исходные файлы веб-сервиса и дескриптор развертывания.
3. Откройте *WSDL*-файл в папке *WebContent\WSDL* и исправьте ссылку на веб-сервис, которая могла быть изменена мастером создания веб-сервисов.
4. Откройте файл реализации веб-сервиса <имя>*Impl.java* и напишите свою реализацию операции вместо шаблонной.

Развертывание веб-сервиса

Веб-сервис будет работать на том же сервере, в котором установлен *BPEL*-контейнер. Это сервер приложений *Apache Tomcat* 5.0, входящий в поставку *ActiveBPEL Designer*.

В контекстном меню проекта в среде *Eclipse* выберите пункт *Export*. Экспортируйте проект в *WAR*-файл.

1. Запустите сервер приложений через скрипт *bin\startup.bat*, находящийся в папке сервера. Появится окно консоли сервера. Об успешном запуске свидетельствует сообщение *INFO: Server startup in 7375 ms* и отсутствие *Java*-исключений.

2. Поместите *WAR*-файл в папку *webapps* на сервере приложений и по сообщениям в консоли сервера и отсутствию сообщений об ошибках убедитесь, что веб-приложение было установлено успешно.

Затем необходимо развернуть веб-сервис. В *Axis* это можно сделать с помощью специального системного веб-сервиса *AdminService*, на вход которому подается дескриптор развертывания *deploy.wsdd*. Для вызова этого сервиса существует готовый клиент. Используйте для развертывания веб-сервиса *bat*-файл следующего содержания, предварительно изменив его под свою систему:

```
set TOMCAT_HOME=G:\Program Files\Active Endpoints\ActiveBPEL Designer\Server\ActiveBPEL_Tomcat (путь установки сервера приложений)

set LIBS=%TOMCAT_HOME%\shared\lib

set SERVLET_URL=http://localhost:8080/<имя-проекта>/services/AdminService



java -cp "%LIBS%\axis.jar";"%LIBS%\commons-discovery.jar";"%LIBS%\commons-logging.jar";"%LIBS%\jaxrpc.jar";"%LIBS%\saaj.jar";"%LIBS%\wsdl4j.jar" org.apache.axis.client.AdminClient -l%SERVLET_URL% deploy.wsdd
```

3. При успешном развертывании сервиса будет выведено сообщение

```
<Admin>Done processing</Admin>
```

4. Проверьте, что сервис был успешно развернут, введя в адресной строке браузера ссылку, указанную в *WSDL*-файле. Должна появиться страница, сообщающая, что по этому адресу установлен *AXIS*-сервис. Добавьте в конце адреса “*?WSDL”* и будет выведено *WSDL*-описание сервиса.
5. Попробуйте выполнить операцию веб-сервиса через *Web Services Explorer* в среде *Eclipse* (*Run → Launch the Web Services Explorer*). Проверьте правильность работы веб-сервиса при различных входных данных (рис. 7).

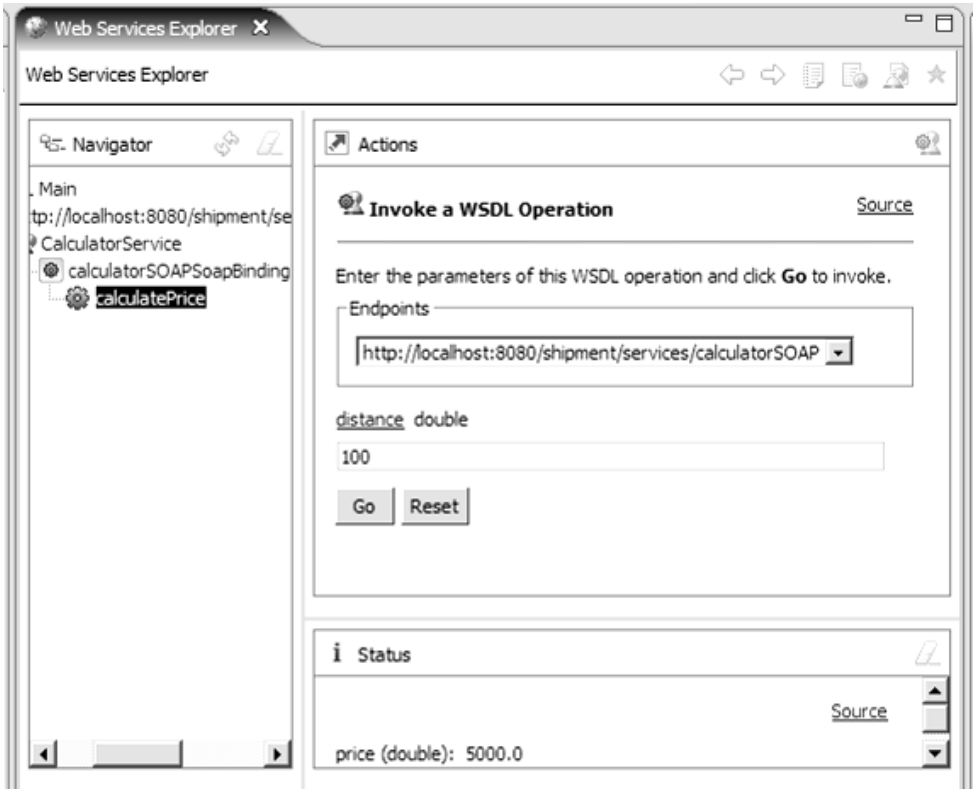

Рис. 7. Испытание веб-сервиса в Web Services Explorer

Развертывание BPEL-процесса

Загрузите *ActiveBPEL Designer* и откройте проект, созданный в предыдущей лабораторной работе.

1. Создайте новый дескриптор развертывания для *BPEL*-процесса (*File → New → Deployment Descriptor*). Укажите файл *BPEL*-процесса. В окне *Partner Links* для *Partner Link*-ов, напротив которых стоят красные крестики, укажите значение *Endpoint Type,* равное *static*, как на рис. 8.

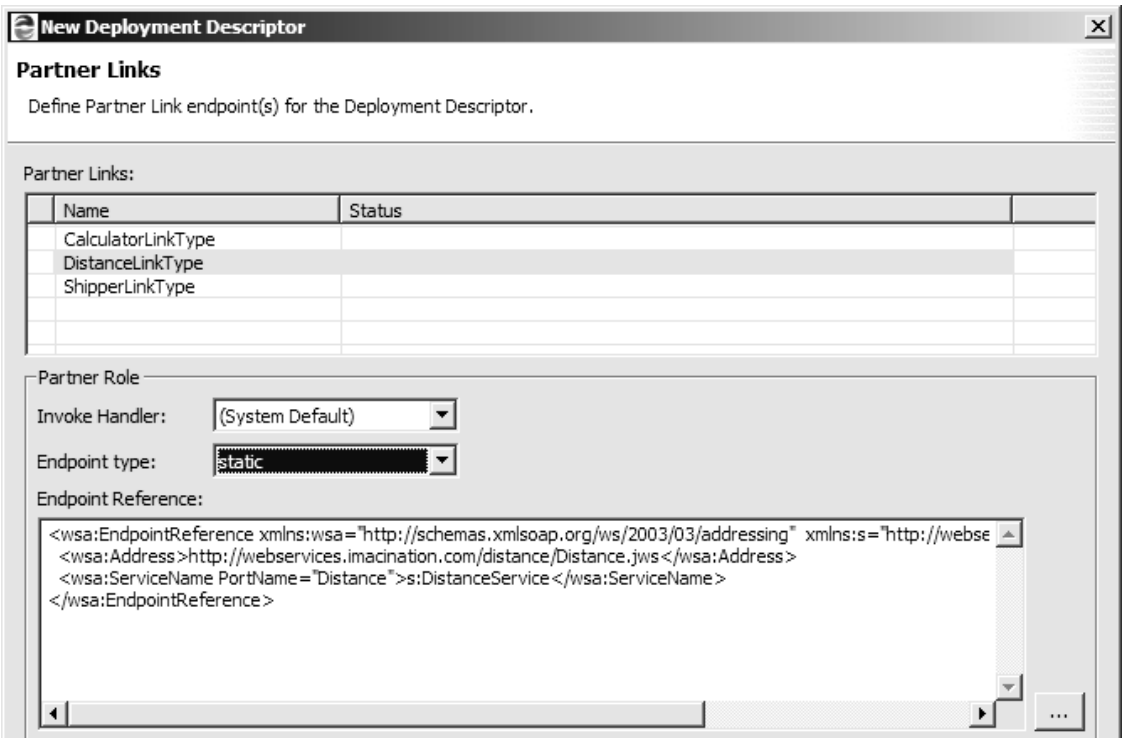

Рис. 8. Создание нового дескриптора развертывания для BPEL-процесса

2. Для *Partner Link*, соответствующего *BPEL*-процессу, установите поле *Binding* в значение *RPC Encoded*. Остальные поля оставьте со значениями по умолчанию. Будет создан дескриптор в файле с расширением *pdd*.

3. Создайте в проекте папку *deploy*, куда потом будут помещаться файлы, связанные с развертыванием (*File* → *New* → *Other* → *Simple* → *Folder*).

4. Создайте *BPR*-файл – архив *BPEL*-процесса, который будет развернут в *BPEL*-контейнере. Для этого в контекстном меню проекта выберите пункт *Export*, *Business Process Archive File*. Пометьте дескриптор развертывания, созданный на предыдущем этапе. В поле *Select the export destination* укажите имя *BPR*-файла в папке *deploy*. Выберите *Deployment Type* → *File* и в поле *Deployment Location* укажите папку *bpr*-сервера приложений. Нажмите *Finish* и *BPEL*-процесс установится в контейнер, при этом в консоли сервера появятся соответствующие сообщения.

Убедиться, что процесс успешно установлен в контейнер можно также через панель администрирования *BPEL Engine*, она находится по адресу

http://localhost:8080/BpelAdmin/. В случае неудачного развертывания посмотрите содержимое *Deployment Log* в панели.

5. После развертывания *BPEL*-процесс доступен для использования как веб-сервис. Вы можете его найти по адресу вида *http://localhost:8080/active-bpel/services/<mun-partner-link-процесса>Service*.

Проверка BPEL-процесса

Проверьте работоспособность созданного *BPEL*-процесса с помощью *Web Services Explorer*. Используйте различные значения переменных, чтобы проверить все пути выполнения (рис. 9).

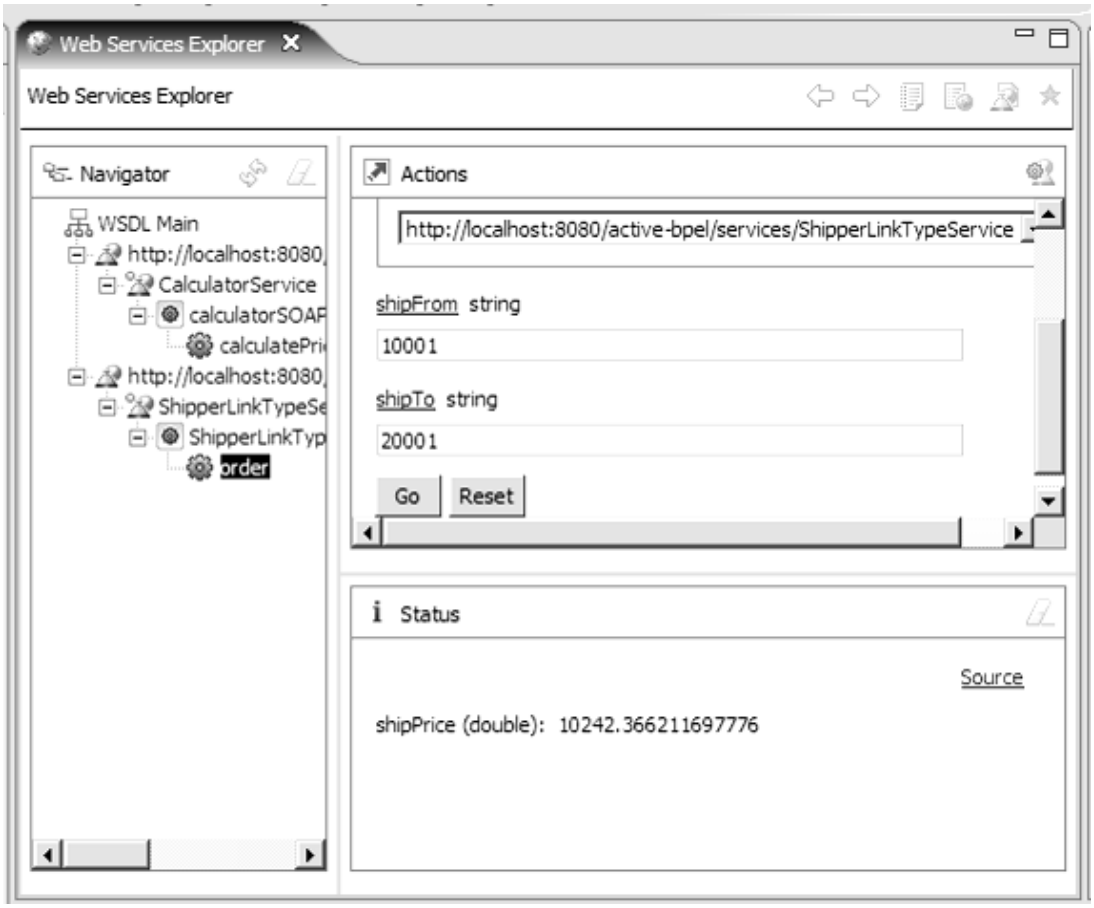

Рис. 9. Результаты выполнения процесса – стоимость доставки контейнера из Нью-Йорка в Вашингтон

Осуществите пошаговое выполнение *BPEL*-процесса. Для этого:

Откройте *BPEL*-процесс в *ActiveBPEL Designer*. Выберите блок *Receive* и установите на нем точку останова (контекстное меню → *Add Breakpoint*).

Создайте конфигурацию для выполнения удаленной сессии отладки. Для этого выберите пункт меню *Run* → *Debug*. Создайте новую конфигурацию, нажав на кнопку *New*. Нажмите кнопку *Debug*, чтобы начать отладку (рис. 10).

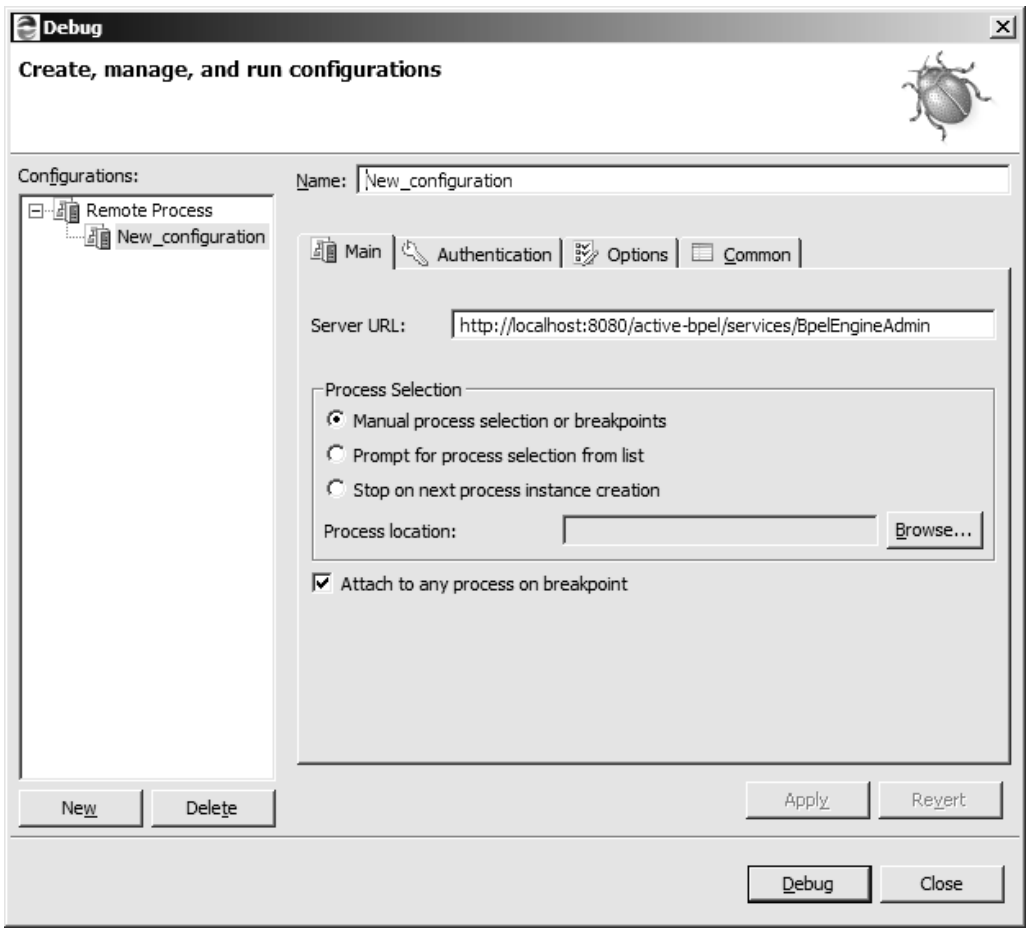

Рис. 10. Конфигурация для удаленной отладки

Запустите сервис *BPEL*-процесса через *Web Services Explorer* и вернитесь в *ActiveBPEL Designer*, и выполните пошагово процесс так же, как при моделировании в предыдущей работе (рис. 11).

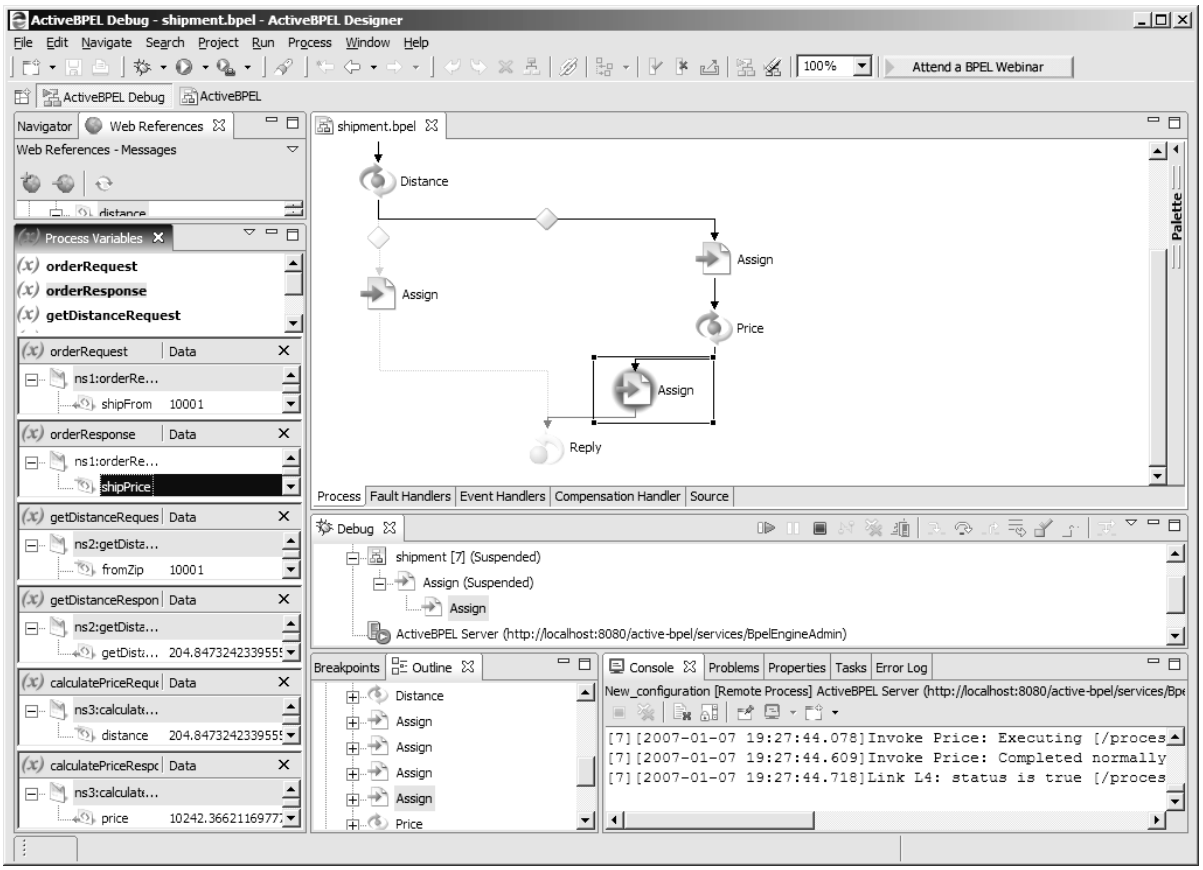

Рис. 11. Сессия удаленной отладки BPEL-процесса

Вопросы для самоконтроля

1. Почему используется контейнер *Apache Axis* для реализации веб-сервисов?
2. Чем «*Top down*» реализация веб-сервиса отличается от «*Bottom up*»?
3. Как определить, что сервер приложений был успешно запущен?
4. Как развернуть веб-сервис в контейнере *Apache Axis*?
5. Перечислите все способы, как убедиться в том, что веб-сервис был успешно развернут в контейнере.
6. Какая информация помещается в дескриптор развертывания *BPEL*-процесса?
7. Какие возможности предоставляет панель администрирования *BPEL Engine*?
8. Перечислите все признаки, по которым можно судить об успешном развертывании *BPEL*-процесса.
9. Как запустить пошаговую удаленную отладку *BPEL*-процесса?

Практическое занятие № 4
Базовые принципы разработки мультиагентных приложений на платформе *JADE*

Цель работы: выполнить простой пример, который продемонстрирует основные шаги разработки мультиагентного приложения на платформе *JADE*:

- Реализация простых агентов
- Компиляция
- Запуск
- Передача параметров
- Поведение агента
- Передача сообщений

Задание

Реализовать сценарий «электронная торговля книгами». В торгах участвуют два типа агентов: агенты, продающие книги, и агенты, покупающие их от лица своих пользователей.

Агент-покупатель запрашивает название интересующей его книги у продавцов. Если книга есть в наличии, происходит сделка, в случае, если книгу предлагают сразу несколько продавцов, покупатель выбирает самое выгодное для себя предложение (самая низкая цена).

Агент-продавец содержит каталог книг и цен на них. При поступлении запроса от покупателя продавец проверяет наличие запрошенной книги. Если книга есть, возвращает ее цену, в противном случае отклоняет запрос. Если клиента устраивает цена, совершается сделка, книга удаляется из каталога продавца.

Предварительные требования

Для выполнения работы на компьютере должно быть установлено следующее ПО:

- *JDK* версии 1.2 или выше;
- платформа *JADE* версии не ниже 1.3.

Порядок выполнения работы

Реализация агентов

Каждый агент платформы *JADE* должен расширять базовый класс *jade.core.Agent*. В простейшем случае агент может быть реализован следующим образом:

```
ЙНРПУФ jade.core.Agent;


РХВМЙГ ГМБ   BookBuyerAgent ЕШФЕОД  Agent {

   РУПФЕГФЕД  ПЙД setup() {

      System.out.println(

         "Hallo! Buyer-agent " + getAID().getName() + " is ready."

      );

   }

}
```

Метод *setup()* служит для реализации действий, которые следует выполнить при инициализации агента.

Все агенты должны иметь уникальный идентификатор, идентификатор реализован с помощью класса *jade.core.AID* и имеет формат :

<имя агента>@<имя платформы>.

Подробная информация о реализации агентов находится в [4].

Компиляция и запуск агента

Созданный агент может быть откомпилирован следующим образом:

```
javac –classpath <JADE-classes> BookBuyerAgent.java

где <JADE-classes> это http.jar, iiop.jar, jade.jar и jadeTools.jar.
```

Для запуска необходимо воспользоваться командой,

```
java -classpath <JADE-classes>;. jade.Boot buyer:BookBuyerAgent ,
```

где *jade.Boot* – системный класс-загрузчик, служит для обработки параметров командной строки; *buyer* – имя агента, реализованного классом *BookBuyerAgent*.

После запуска в окне консоли должны появиться системные сообщения о готовности платформы и сообщение-приветствие от агента:

```
Hallo! Buyer-agent a@ibm:1099/JADE is ready.
```

Передача параметров агенту

В базовом классе агентов реализован метод *getArguments*(), который возвращает аргументы командной строки в виде массива объектов класса *Object*. Воспользуемся этим способом для передачи агенту-покупателю название книги для поиска :

```
ЙНРПУФ jade.core.AID;

ЙНРПУФ jade.core.Agent;


РХВМЙГ ГМБ  BookBuyerAgent ЕШФЕОД  Agent {


   РУЙ БФЕ String targetBookTitle;
   РУЙ БФЕ AID[] sellerAgents = {
      ОЕ  AID("seller1", AID.ISLOCALNAME),
      ОЕ  AID("seller2", AID.ISLOCALNAME)
   };


   РУПФЕГФЕД  ПЙД setup() {


      System.out.println(
         "Hallo! Buyer-agent " + getAID().getName()+ " is ready."
      );
```

```
      Object[] БУЗ  = getArguments();
      ЙЖ (БУЗ  != ОХММ && БУЗ .length > 0) {
         targetBookTitle = (String) БУЗ [0];
         System.out.println("Trying to buy " + targetBookTitle);
      } ЕМ Е {

         System.out.println("No book title specified");
         doDelete();
      }
   }

   РУПФЕГФЕД  ПЙД takeDown() {
      System.out.println(
         "Buyer-agent " + getAID().getName() + " terminating."
      );
   }
}
```

Аргументы командной строки должны быть заключены в скобки и разделены пробелами :

```
java jade.Boot buyer:BookBuyerAgent(The-Lord-of-the-rings)
```

Должно появиться такое сообщение:

```
Hallo! Buyer-agent buyer@ibm:1099/JADE is ready.

Trying to buy The-Lord-of-the-rings
```

Поведение агента – класс Behaviour

Все задачи, которые агент должен выполнять, будем называть поведением агента. Поведение агента реализуется классом, расширяющим абстрактный базовый класс *jade.core.behaviours.Behaviour*. Для добавления по-

ведения агенту можно воспользоваться методом *addBehaviour*() класса *jade.core.Agent*. Поведение может быть добавлено в любой момент жизненного цикла агента (при инициализации агента или в ходе выполнения какой-то задачи).

Класс, расширяющий *jade.core.behaviours.Behaviour*, должен реализовывать два абстрактных метода *action*() и *done*(). Метод *action*() реализует функциональность поведения, *done*() проверяет, завершилось ли выполнение задачи и возвращает *true* или *false* :

```
РХВМЙГ ГМБ  OverbearingBehaviour ЕШФЕОД  Behaviour {

   РХВМЙГ  ПЙД action() {

       ИЙМЕ (ФУХЕ) {

         // do something

      }

   }


   РХВМЙГ ВППМЕБО done() {

      return true;

   }

}
```

Агент может содержать несколько вариантов поведения. Поведения выполняются последовательно (в противоположность потокам), переключение между различными вариантами поведения контролирует разработчик.

Полную информацию о реализации поведения агентов можно найти в [4].

Поведение агентов-покупателей

Агент-покупатель периодически опрашивает продавцов на предмет наличия книги. Поведение, характеризующееся периодичностью выполнения однотипных действий, удобно реализовать, расширяя не стандартный ба-

зовый класс, а класс *jade.core.behaviours.TickerBehaviour*. В этом случае необходимо реализовать абстрактный метод *onTick*() и задать период его повторений. Модифицируем метод *setup*() агента :

```
protected void setup() {

      System.out.println(

         "Hallo! Buyer-agent " + getAID().getName() + " is ready."

      );


      Object[] БУЗ  = getArguments();

      ЙЖ (БУЗ  != ОХММ && БУЗ .length > 0) {

         targetBookTitle = (String) БУЗ [0];

         System.out.println("Trying to buy " + targetBookTitle);

         addBehaviour(ОЕ  TickerBehaviour(ФИЙ , 60000) {

            РУПФЕГФЕД  ПЙД onTick() {

               myAgent.addBehaviour(ОЕ  RequestPerformer());

            }

         });

      } ЕМ Е {

         System.out.println("No target book title specified");

         doDelete();

      }

   }
```

Реализация *RequestPerformer* будет рассмотрена позднее после изучения взаимодействия агентов.

Поведение агентов-продавцов

Агент-продавец должен реализовать следующие варианты поведения.

Прием заявок от покупателей (*OfferRequestsServer*).

Оформление сделки (*PurchaseOrdersServer*).

Добавление книги в каталог.
Первые два рассмотрим после изучения взаимодействия агентов.
Класс агента-продавца :

```
ЙНРПУФ jade.core.Agent;

ЙНРПУФ jade.core.behaviours.OneShotBehaviour;

ЙНРПУФ java.util.Hashtable;


РХВМЙГ ГМБ  BookSellerAgent ЕШФЕОД  Agent {


   РУЙ БФЕ Hashtable catalogue;


   РУЙ БФЕ BookSellerGui myGui;


   РУПФЕГФЕД  ПЙД setup() {
      catalogue = ОЕ  Hashtable();
      myGui = ОЕ  BookSellerGui(ФИЙ );
      myGui.show();
      addBehaviour(ОЕ  OfferRequestsServer());
      addBehaviour(ОЕ  PurchaseOrdersServer());
   }



   РУПФЕГФЕД  ПЙД takeDown() {
      myGui.dispose();
      System.out.println(
         "Seller-agent " + getAID().getName() + " terminating."
      );
   }
```

```
   РХВМЙГ  ПЙД updateCatalogue(ЖЙОБМ String ФЙФМЕ, ЖЙОБМ ЙОФ РУЙГЕ) {
      addBehaviour(ОЕ  OneShotBehaviour() {
         РХВМЙГ  ПЙД action() {
            catalogue.put(ФЙФМЕ, ОЕ  Integer(РУЙГЕ));
         }
      });
   }
}
```

Взаимодействие агентов

Платформа *JADE* для передачи сообщений между агентами использует язык *ACL*.

Каждое сообщение реализуется объектом класса *jade.lang.acl.ACLMessage.*

Для отправки сообщения необходимо заполнить поля объекта сообщения и передать его методу *send*() класса *jade.core.Agent* :

```
ACLMessage Н З = ОЕ  ACLMessage(ACLMessage.INFORM);
Н З.addReceiver(ОЕ  AID("Peter", AID.ISLOCALNAME));
Н З.setLanguage("English");
Н З.setOntology("Weather-forecast-ontology");
Н З.setContent("Today it's raining");
send(Н З);
```

Отправленные сообщения платформа *JADE* автоматически ставит в очередь доставленных сообщений агента-получателя. Агент может получить к ним доступ, вызвав метод *receive*(), который возвращает первое сообщение из очереди или *null,* если очередь пустая :

```
ACLMessage Н З = receive();
ЙЖ (Н З != ОХММ) {
```

```
// Обработка сообщения

}
```

Часто необходимо реализовать поведение агента, который обрабатывает сообщения от других агентов. В нашем случае это *OfferRequestsServer and PurchaseOrdersServer* – в каждом из них требуется обработка заявок покупателя. Такое поведение должно быть повторяющимся, а при каждой итерации необходимо проверять наличие новых сообщений и обрабатывать их. Методы аналогичны, рассмотрим реализацию *OfferRequestsServer*.

Реализация поведения агента-продавца – обработка заявки :

```
РХВМЙГ ГМБ  OfferRequestsServer ЕШФЕОД  CyclicBehaviour {

   РХВМЙГ  ПЙД action() {

      ACLMessage Н З = myAgent.receive();

      ЙЖ (Н З != ОХММ) {


         //Сообщение принято, обработаем его


         String ФЙФМЕ = Н З.getContent();

         ACLMessage УЕРМЩ = Н З.createReply();

         Integer РУЙГЕ = (Integer) catalogue.get(ФЙФМЕ);

         ЙЖ (РУЙГЕ != ОХММ) {


            // Запрашиваемая книга есть у продавца. Возвращаем ее цену


            УЕРМЩ.setPerformative(ACLMessage.PROPOSE);

            УЕРМЩ.setContent(String.valueOf(РУЙГЕ.intValue()));

         } ЕМ Е {


            // Запрашиваемая книга отсутствует у продавца
```

```
            УЕРМЩ.setPerformative(ACLMessage.REFUSE);
            УЕРМЩ.setContent("not-available");
          }
          myAgent.send(УЕРМЩ);
       }
   }
}
```

Выбор сообщений с заданными характеристиками

Реализованные поведения *OfferRequestsServer* и *PurchaseOrdersServer* последовательно выбирают из очереди новые сообщения для обработки, при этом *OfferRequestsServer* обрабатывает сообщения с запросом на книгу, а *PurchaseOrdersServer* – сообщения с предложением цены. Для того чтобы различать эти сообщения в очереди, необходимо воспользоваться так называемыми шаблонами. Шаблон будет использоваться методом *receive*() для выбора сообщений, удовлетворяющих этому шаблону. Шаблоны реализуются классом *jade.lang.acl.MessageTemplate*, он содержит методы для манипулирования шаблонами. В метод *action*() необходимо внести соответствующие изменения:

```
      РХВМЙГ  ПЙД action() {
            MessageTemplate mt = MessageTemplate.MatchPerformative(
  ACLMessage.CFP
);
            ACLMessage msg = myAgent.receive(mt);
            ЙЖ (msg != ОХММ) {
               ...
            } ЕМ Е {
                  block();
            }
      }
```

Подробную информацию о реализации *ACL* можно найти в [4].

Система переговоров агентов

Переговорами будем называть последовательность сообщений между двумя или более агентами, связанных по смыслу. Система переговоров присутствует в поведении, реализованном классом *RequesrPerformer*. Для контроля стадии переговоров удобно внедрять в сообщения специальные идентификаторы :

```
import jade.core.AID;
import jade.core.behaviours.Behaviour;
import jade.lang.acl.ACLMessage;
import jade.lang.acl.MessageTemplate;


public class RequestPerformer extends Behaviour {
   private AID bestSeller; // агент-продавец с лучшей(наименьшей ценой)
   private int bestPrice; // лучшая цена
   private int repliesCnt = 0; // счетчик ответов от продавцов
   private MessageTemplate mt;
   private int step = 0;


   public void action() {
      switch (step) {
      case 0:
         // отправим запрос на книгу всем продавцам
         ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
         for (int i = 0; i < sellerAgents.length; ++i) {
            cfp.addReceiver(sellerAgents[i]);
         }
         cfp.setContent(targetBookTitle);
         cfp.setConversationId("book-trade");
```

```
            cfp.setReplyWith("cfp" + System.currentTimeMillis());
            myAgent.send(cfp);
            // подготовим шаблон для получения цен на книги
            mt = MessageTemplate.and(MessageTemplate
               .MatchConversationId("book-trade"), MessageTemplate
               .MatchInReplyTo(cfp.getReplyWith()));
            step = 1;
            break;
         case 1:
            // получим ответы с ценами, либо отказами, если книги нет
            ACLMessage reply = myAgent.receive(mt);
            if (reply != null) {
               if (reply.getPerformative() == ACLMessage.PROPOSE) {
                  int price = Integer.parseInt(reply.getContent());
                  if (bestSeller == null || price < bestPrice) {
                     // это лучшее предложение
                     bestPrice = price;
                     bestSeller = reply.getSender();
                  }
               }
               repliesCnt++;
               if (repliesCnt >= sellerAgents.length) {
                  // приняты ответы от всех продавцов
                  step = 2;
               }
            } else {
               block();
            }
            break;
```

```
      case 2:


         // отправим запрос на покупку агенту, предложившему лучшую цену
         ACLMessage order = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
         order.addReceiver(bestSeller);
         order.setContent(targetBookTitle);
         order.setConversationId("book-trade");
         order.setReplyWith("order" + System.currentTimeMillis());
         myAgent.send(order);
         mt = MessageTemplate.and(MessageTemplate
            .MatchConversationId("book-trade"), MessageTemplate
            .MatchInReplyTo(order.getReplyWith()));
         step = 3;
         break;
      case 3:
         reply = myAgent.receive(mt);
         if (reply != null) {
            // принят ответ не предложение покупки
            if (reply.getPerformative() == ACLMessage.INFORM) {
               // покупка совершена
               System.out.println(targetBookTitle    +    "    successfully
purchased.");
               System.out.println("Price = " + bestPrice);
               myAgent.doDelete();
            }
            step = 4;
         } else {
            block();
         }
```

```
            break;

        }

    }


    public boolean done() {

        return ((step == 2 && bestSeller == null) || step == 4);

    }

}
```

Обычно переговоры проходят по стандартной, заранее известной, схеме (протоколу). *JADE* предоставляет возможность реализовать переговоры по протоколу с помощью классов из пакета *jade.proto*.

Подробную информацию об использовании протокола взаимодействия можно найти в [4].

Получение сообщений в режиме блокировки

Кроме метода *receive*() класс *Agent* предоставляет еще один метод для получения входящих сообщений – метод *blockingReceive*(). Отличие этого метода заключается в том, что возврат из него не происходит пока не будет получено сообщение, то есть происходит блокировка агента (потока агента и, следовательно, его поведения). Аналогично, перегруженная версия метода *blockingReceive*, принимающая в качестве параметра шаблон, блокируется до получения сообщения, удовлетворяющего используемому шаблону.

Сервис «Желтые страницы»

Сервис «желтые страницы» позволяет агентам публиковать сервисы, которые они предоставляют, и находить сервисы, необходимые для функционирования. Сервис «желтые страницы» реализован в *JADE* (в соответствии со спецификацией *FIPA*) агентом *DF* (*Directory Facilitator*). Взаимодействие с этим агентом может происходить стандартным образом – с помощью сообщений *ACL*. Однако для упрощения этой процедуры *JADE* предоставляет специальный класс *jade.domain.DFService,* с помощью которого можно публиковать и искать сервисы.

Для того чтобы агент мог опубликовать свой сервис, он должен предоставить *DF* следующую информацию.

Описание агента (*AID*, Список языков взаимодействия (опционально), Список онтологий (опционально), Список сервисов).

Описание сервисов (Тип, Имя, Список языков взаимодействия (опционально), Список онтологий (опционально)).

Список свойств (опционально).

Для предоставления этой информации необходимо воспользоваться соответствующими классами из пакета *jade.domain.FIPAAgentManagement*: *DFAgentDescription*, *ServiceDescription* и *Property*. Приведем пример публикации сервиса агента-продавца :

```
protected void setup() {

   ...

   // публикуем сервис агента-продавца
   DFAgentDescription dfd = new DFAgentDescription();
   dfd.setName(getAID());
   ServiceDescription sd = new ServiceDescription();
   sd.setType("book-selling");
   sd.setName("JADE-book-trading");
   dfd.addServices(sd);
   try {
     DFService.register(this, dfd);
   }
   catch (FIPAException fe) {
     fe.printStackTrace();
   }
```

При завершении агента принято удалять сервис :

```
РУПФЕГФЕД  ПЙД takeDown() {
  ФУЩ {
    DFService.deregister(ФИЙ );
  }
  ГБФГИ (FIPAException ЖЕ) {
    ЖЕ.printStackTrace();
  }
    myGui.dispose();
  System.out.println("Seller-agent "+getAID().getName()+" terminating.");
}
```

Поиск сервиса происходит аналогично с помощью тех же классов :

```
  DFAgentDescription ФЕНРМБФЕ = ОЕ  DFAgentDescription();
  ServiceDescription  Д = ОЕ  ServiceDescription();
   Д.setType("book-selling");
  ФЕНРМБФЕ.addServices( Д);
  ФУЩ {
     DFAgentDescription[] УЕ ХМФ = DFService.search(myAgent, ФЕНРМБФЕ);
     System.out.println("Found the following seller agents:");
     sellerAgents = ОЕ  AID[УЕ ХМФ.length];
     ЖПУ (ЙОФ Й = 0; Й < УЕ ХМФ.length; ++Й) {
         sellerAgents[Й] = УЕ ХМФ[Й].getName();
         System.out.println(sellerAgents[Й].getName());
       }
     }
     ГБФГИ (FIPAException ЖЕ) {
```

```
            FE.printStackTrace();

        }
```

JADE предоставляет графический интерфейс к агенту *DF*, в документации к платформе содержится руководство по его использованию.

Диаграмма классов разрабатываемой системы представлена на рис. 12.

Рис. 12. Диаграмма классов

Запуск приложения

Запуск агентов будем производить с помощью графического интерфейса, который предоставляет платформа. Для этого воспользуемся командой

```
java  jade.Boot -gui
```

Руководство по работе в графической среде можно найти в [5].

Добавим двух агентов-продавцов и одного агента-покупателя.

Продавец №1

Agent Name – seller1

Agent Class – examples.bookTrading.BookSellerAgent

В появившемся окне необходимо ввести название книги и ее цену для формирования каталога продавца.

Book title – TheLordOfTheRings

Price – 100

Продавец №2

Agent Name – seller2

Agent Class – examples.bookTrading.BookSellerAgent

Каталог продавца:

Book title – TheLordOfTheRings

Price – 115

Покупатель

Agent Name – buyer

Agent Class – examples.bookTrading.BookBuyerAgent

Arguments – TheLordOfTheRings

За ходом взаимодействия агентов можно наблюдать в окне консоли. Более полное представление о взаимодействии агентов дает утилита *sniffer*, руководство по работе со сниффером можно найти в документации к платформе (рис. 13).

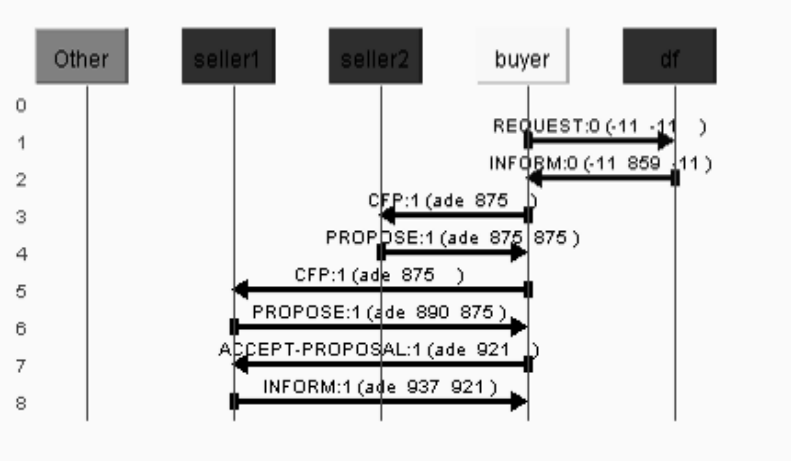

Рис. 13. Обмен сообщениями между агентами

В результате переговоров покупатель совершил сделку с продавцом *seller1*, выбрав книгу по более выгодной для себя цене – 100, в то время как *seller2* предлагал ее за 115.

Варианты индивидуальных заданий

Продумать и реализовать денежные операции между продавцом и покупателем.

Реализовать переговоры агентов с использованием протокола.

Вопросы для самоконтроля

1. Какие требования предъявляются к реализации агента?
2. В чем заключаются особенности реализации поведения агентов. Как происходит переключение между вариантами поведения?
3. Какой язык используется для передачи сообщений между агентами?
4. Как осуществляются переговоры между агентами?
5. В чем сущность получения сообщений в режиме блокировки?
6. Каково предназначение и как реализован сервис «Желтые страницы»?

Практическое занятие №5
Работа с онтологиями

Цель работы: продемонстрировать принципы работы с онтологиями в рамках платформы *JADE*.

Задание

Реализовать простую онтологию, относящуюся к предметной области, – отдел кадров.

Предварительные требования

Для выполнения работы на компьютере должно быть установлено следующее ПО:

- *JDK* версии 1.2 или выше;
- платформа *JADE* версии не ниже 1.3.

Порядок выполнения работы

Онтология используется двумя агентами – *RequesterAgent* (претендент на должность), *EngagerAgent* (отдел кадров).

Реализация онтологии

Онтология состоит из следующих сущностей:

Концепты (Человек (*PERSON*), Компания (*COMPANY*), Адрес (*ADDRESS*));

Предикаты (Работает-на (*WORKS-FOR*) – определяет факт того, что Человек работает в Компании);

Действия (Нанять (*ENGAGE*) – Нанять Человека в Компанию);

Предложения (*ENGAGEMENT-ERROR* – ошибка, произошедшая во время выполнения действия *ENGAGE*; *PERSON-TOO-OLD* – Человек слишком старый, чтобы работать).

В каталоге *employment* содержатся классы, реализующие сущности онтологии. Класс *EmploymentOntology* создает онтологию и добавляет в нее объекты классов, реализующих сущности.

Для работы с онтологиями используются классы пакета *jade.content*. Так, для реализации сущностей онтологии – Концепты, Предикаты, Дей-

ствия – необходимо реализовать интерфейсы *jade.content.Concept*, *jade.content.Predicate*, *jade.content.AgentAction* соответственно.

Сущности онтологии представляют собой классы *java beans*, реализующие один из интерфейсов в зависимости от своего назначения (рис. 14).

Так, например, концепт Компания должен быть реализован следующим образом:

```
РБГЛБЗЕ examples.ontology.employment;


ЙНРПУФ jade.content.Concept;


public class Company implements Concept {


      РУЙ БФЕ String    _name; // Название компании
      РУЙ БФЕ Address   _address; //Адрес


      // getter/setter методы, которые будут вызываться платформой
      РХВМЙГ  ПЙД setName(String ОБНЕ) {
            _name=ОБНЕ;
      }
      РХВМЙГ String getName() {
            УЕФХУО _name;
      }
      РХВМЙГ  ПЙД setAddress(Address БДДУЕ  ) {
            _address=БДДУЕ  ;
      }
      РХВМЙГ Address getAddress() {
            УЕФХУО _address;
      }
```

```
        // методы, специфичные для приложения

        РХВМЙГ ВППМЕБО equals(Company Г){

                УЕФХУО (_name.equalsIgnoreCase(Г.getName()));

        }

}
```

Рис. 14. Диаграмма классов онтологии

RequesterAgent

Агент, представляющий претендента на должность, получает сведения о кандидате и предоставляет их агенту-нанимателю *EngagerAgent*. Агент имеет один тип поведения, действия которого выполняются циклически:

- получить сведения от человека, реализовано в методе *onStart*().
- послать запрос агенту *EngagerAgent* с целью проверки, что кандидат не работает в компании. Реализуется классом *SimpleAchieveREInitiator* в соответствии с протоколом *FIPA-Query*. Класс *SimpleAchieveREInitiator* добавлен как дополнительный класс поведения.

В зависимости от результата запроса человека принимают в компанию. В методе *onEnd*() происходит возврат к началу цикла поведения.

EngagerAgent

Агент-наниматель выполняет функции отдела кадров. Агент имеет два типа поведения.

SimpleAchieveREResponder – обрабатывает запросы о людях, работающих в компании.

Еще один *SimpleAchieveREResponder* – обрабатывает запросы на работу.

Оба поведения циклические.

Список сотрудников, уже работающих в компании, является объектом класса, реализующего интерфейс *jade.util.leap.List* – это зависимый от окружения аналог *java.util.List*.

Запуск приложения

Запуск агентов будем производить с помощью графического интерфейса, который предоставляет платформа. Для этого воспользуемся командой

```
java  jade.Boot -gui
```

Руководство по работе в графической среде можно найти в [5].

Добавим агента, представляющего организацию,

Agent Name – engager

Agent Class – examples.ontology.EngagerAgent.

После добавления агента в окне консоли должно появиться следующее сообщение:

```
This is the EngagerAgent representing the company CSELT.
```

Агент готов принимать запросы на должности в компании. Имя компании фиксировано – *CSELT*, это необходимо учесть при запуске агентов, представляющих интересы претендентов на должность.

Добавим агента, представляющего кандидата,
Agent Name – requester
Agent Class – examples.ontology.RequesterAgent.
После добавления агента в окне консоли должно появиться предложение ввести имя агента-нанимателя:

```
ENTER the local name of the Engager agent -->
```

Необходимо ввести – *engager*.
Теперь требуется ввести информацию о компании, в которую нанимаются претенденты:

```
ENTER details of the company where people will be engaged

  Company name --> CSELT

  Company address

    Street ------> street

    Number ------> 1

    City   ------> city
```

В поле *Company name* обязательно ввести *CSELT*.

Наконец, можно ввести данные о претенденте:

```
ENTER details of person to engage

  Person name --> Name

  Person age ---> 25

  Person address

    Street -----> street

    Number -----> 1

    City   -----> city

Engagement agreed. Waiting for completion notification...
```

```
Engagement successfully completed
```

Вводя различную информацию, следует проверить различные прецеденты в работе агента-представителя компании.

Варианты индивидуальных заданий

Реализовать возможность представлять различные компании.

Реализовать возможность менять максимальный возраст для претендентов.

Вопросы для самоконтроля

1. Что такое онтология?
2. Из каких сущностей состоит онтология, в чем их отличия?
3. Какие средства предоставляет платформа *JADE* для реализации онтологии и ее сущностей?

Практическое занятие № 6
Организация групповой работы в среде *Windows Sharepoint Services 2003*

Цель работы: изучить теоретические основы технологии *groupware*, ее цели и задачи. Познакомиться с решением *Microsoft* в области организации групповой работы на примере технологии *Windows SharePoint Services*. Выполнить развертывание и настройку среды групповой работы в соответствии с методическими указаниями.

Теоретические сведения по технологии *GROUPWARE*

Быстрое развитие веб-технологий и технологий передачи данных связано с потребностью в рабочей среде круглосуточного действия, необходимого членам групп для тесного сотрудничества, решения задач и достижения целей масштаба группы и организации. Несмотря на то что распределенные рабочие коллективы – неотъемлемая часть современного бизнеса, эффективную совместную работу часто бывает трудно обеспечить.

Каждая информационная технология поддерживает определенные методы управления. Метод управления определяет то, на что и как надо воздействовать управляющему для достижения ожидаемых результатов бизнеса. Рассмотрим наиболее распространенные методы, которые в менеджменте можно назвать как управленческие стандарты:

- Управление ресурсами. Модель этих методов представляет организацию как систему ресурсов (финансов, материальных запасов, кадров), принадлежащих владельцам – юридическим лицам, структурным подразделениям, физическим лицам. Все процессы описываются как проводки, отражающие перемещение ресурсов между владельцами. Метод управления хорошо описывается моделями, ставшими стандартами: модель бухгалтерского учета (например *GAAP*), планирование производственных ресурсов (*MRP II*), планирование всех ресурсов предприятия (*ERP*);
- Управление процессами. Эта группа методов представляет организацию как систему бизнес-процессов. Здесь центральными понятиями

выступают процесс, функция, данные, событие. Основная цель управления для этих методов – обеспечение координации событий и функций. К этой группе можно отнести такие методы, как управление качеством (*TQM* – стандарт *ISO9000*), управление процессами (*Workflow* – стандарты ассоциации *Workflow Management Coalition*), а также управление проектами (семейство стандартов *PMI*), но лишь в той степени, в какой эти проекты можно считать типовыми, сведенными до уровня технологии;

- Управление коммуникациями и знаниями. Эти методы представляют организацию как систему небольших коллективов сотрудников, решающих общую задачу, а в роли организующих факторов выступают корпоративные знания и эффективные коммуникации. Главным корпоративным ресурсом управления становится база корпоративных знаний, в которой сотрудники могут быстро найти информацию для принятия правильного решения и понимания друг друга. Эта база концентрирует в себе коллективный опыт компании и создает контекст корпоративных коммуникаций. Основная цель управления – обеспечение координации, коммуникации и быстрого поиска информации для самостоятельного принятия решения. К этой группе методов также относятся методы управления сложными нестандартными проектами (семейство стандартов *PMI*). В таких проектах критическим фактором управления являются проектные коммуникации и квалификационный уровень проектной группы.

Такие технологии, как порталы, рабочие области групп, электронная почта, уведомления о присутствии, мгновенные сообщения и веб-конференции составляют портфель служб для совместной работы и взаимодействия, позволяющие связать людей, данные, процессы и системы внутри и вне корпоративного брандмауэра. Эти ИТ-ресурсы удовлетворяют следующим ключевым требованиям сотрудников организаций:

- Простой поиск данных
- Повышение производительности работы групп
- Тесное и эффективное взаимодействие с коллегами
- Удобство работы пользователя

Groupware – система поддержки коллективной работы. Программы класса *groupware* облегчают специалистам и менеджерам обмен сообщениями и документами, но в отличие от систем класса *workflow* не привязаны к конкретному бизнес-процессу или проекту. Средства организации совместной работы позволяют наладить взаимодействие в случаях, не поддающихся строгой формализации. Все, что относится к творческой работе и где присутствует переговорный процесс, требует гибкого подхода, поэтому клиентов, поставщиков и партнеров никто не включает во внутренние бизнес-процессы организации. Технология *groupware* позволяет управлять отдельными рабочими группами в составе крупных проектов, одновременно обеспечивая необходимый уровень интеграции данных и изоляции процессов их обработки.

Теоретические сведения по технологии *Windows SharePoint Services*

Windows SharePoint Services – это набор служб для *Microsoft Windows Server 2003*, которые предоставляют возможности совместного использования данных, коллективной работы пользователей над документами и создания списков и страниц веб-компонентов. Кроме того, службы *Windows SharePoint Services* можно использовать в качестве базовой платформы создания приложений для совместной работы и использования данных.

Узлы *SharePoint* создаются на основе веб-компонентов и средств *Windows ASP.NET*. Веб-компоненты предназначены для добавления на страницы и настройки администраторами и пользователями узлов и позволяют создать на основе страниц готовые приложения.

Приложения *Microsoft Office System* используют содержимое узла *SharePoint*. Просмотр и редактирование всех представленных на узле совместных материалов, включая документы, списки, события, назначения задач, списки членов, можно выполнять непосредственно в приложениях *Microsoft Office Word 2003*, *Microsoft Office Excel 2003* и *Microsoft Office PowerPoint 2003*. Кроме того, возможно редактирование рисунков из веб-библиотек фотографий. Приложение *Microsoft Office Outlook 2003* позволяет просматривать наряду с личными календарями календари событий уз-

лов *SharePoint*, а также создавать рабочие области для собраний расширенных групп.

Microsoft Office SharePoint Portal Server 2003 – это масштабируемое серверное решение для построения портала организации, основанное на службах *Windows SharePoint Services*, которое можно использовать для объединения узлов *SharePoint*, данных и приложений в организации в рамках единого, простого в использовании портала. Помимо возможностей *Windows SharePoint Services SharePoint Portal Server 2003* предоставляет следующие:

- Новости и темы
- Персональные узлы с личными и общими представлениями
- Адресация информации конкретной аудитории
- Мощные возможности индексирования и поиска в общих файлах, на веб-серверах, защищенных веб-серверах, в общедоступных папках *Microsoft Exchange Server*, *Lotus Notes* и на узлах *SharePoint*.
- Оповещения об изменениях интересующей информации, документов или приложений
- Единый вход в систему для интеграции приложений организации

Поскольку для *SharePoint Portal Server 2003* требуется наличие служб *Windows SharePoint Services*, в *SharePoint Portal Server 2003* доступны все возможности *Windows SharePoint Services*.

Задание

Создать информационный узел на примере вашей кафедры. Необходимо реализовать несколько библиотек документов, связанные настраиваемые списки, отображающие примерную структуру различной кафедральной информации, разработать план событий кафедры и организовать соответствующие рабочие области для событий.

Предварительные требования

Для выполнения работы на компьютере должно быть установлено следующее ПО:

- установленная ОС *Microsoft Windows 2003*;
- установочный файл *WSPS STSV2.EXE*.

Порядок выполнения работы

Пример развертывания служб Windows Sharepoint Services

В качестве учебного примера рассмотрим задачу создания информационного узла для совместной работы одной из кафедр университета на базе *Windows SharePoint Services (WSPS)*. В лабораторной работе будет уделено внимание следующим возможностям служб *WSPS*:

- Общие сведения о среде *WSPS*
- Работа с общими документами на узле, интеграция с *Microsoft Office System*
- Управление задачами и планирование собраний, узлы собраний
- Создание списков произвольного содержания
- Настройка элементов узла под пользователя, веб-части (*web-parts*) и компоненты

Установка Windows SharePoint Services

- Существуют две конфигурации *Windows SharePoint Services*: изолированный сервер и ферма серверов. Первая из них рекомендуется на тот случай, если планируется незначительный уровень использования узлов *SharePoint*. Если же необходимо организовать поддержку веб-узлов в большой организации или компании, выступающей в качестве поставщика услуг Интернета, и предполагаются очень высокий уровень нагрузки и большой объем данных, следует использовать ферму серверов. В данной работе рассмотрены вопросы установки и настройки *Windows SharePoint Services* на изолированном сервере с *Windows SharePoint Services* и модулем *Microsoft SQL Server 2000 Desktop Engine* (*WMSDE*). Предполагается, что на изолированном сервере установлен новый экземпляр *Microsoft Windows Server 2003*.
- Когда программа установки выполняется со стандартными параметрами, виртуальный веб-сервер по умолчанию («Веб-узел по умолчанию» в *IIS*) расширяется за счет возможностей *Windows SharePoint Services*. Если на веб-узле по умолчанию в *IIS* уже настроен определенный веб-узел, то *Windows SharePoint Services* занимает его в процессе установки.

- Для установки *WSPS* на сервер с установленной операционной системой *Microsoft Windows 2003* и настроенным на работу веб-сервером *IIS* необходимо выполнить следующие действия:

1. Запустить установочный файл *STSV2.EXE*;
2. Выбрать обычный тип установки;
3. Дождаться выполнения установки *WSPS* и завершения автоматической настройки веб-сервера *IIS* (рис. 15).

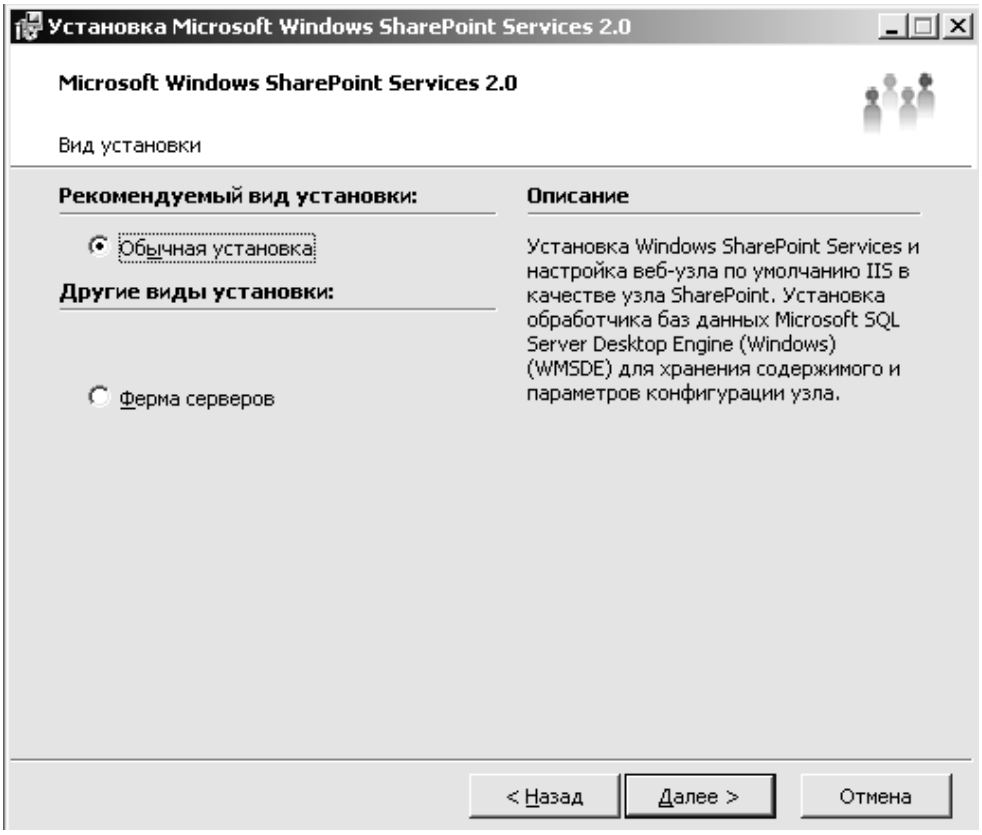

Рис. 15. Установка службы WSPS

После установки службы *WSPS* в разделе администрирования «Панели управления» появится ярлык настройки конфигурации *WSPS*, а при обращении по локальному адресу *http://localhost/* будет доступна «Домашняя страница» узла *WSPS*.

Также рекомендуется установить пакет обновлений для служб *WSPS* и дополнительные веб-части и компоненты *Windows Office System*.

Основные объекты среды WSPS

Домашняя страница узла *WSPS* после установки имеет вид, представленный на рис. 16.

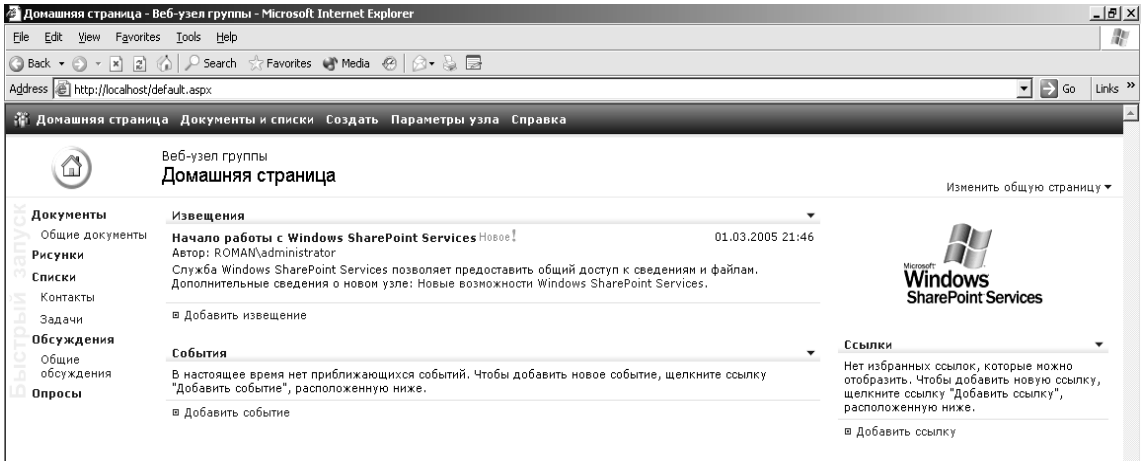

Рис. 16. Вид «домашней страницы» узла WSPS

Среда *WSPS* позволяет создавать различные виды содержимого узла, полный перечень и краткое описание которых можно получить, выбрав в меню узла пункт *«Создать»*. Стандартный список типов содержимого может быть расширен за счет создания собственных шаблонов путем настройки стандартных типов содержимого.

Все содержание узлов *WSPS (контент)* объединяется в *веб-части*, настраивая которые можно получать оптимальное представление данных на узле. Возможны варианты создания самостоятельных *узлов* и *рабочих областей* для проведения собраний и коллективной работы над документами.

Следует обратить внимание, что при создании нового содержания на каком-либо узле или в рабочей области эта информация остается изолированной в рамках узла или рабочей области. Этот факт является одновременно преимуществом (информация при работе разных рабочих групп или над разными проектами не смешивается) и недостатком (получение обобщенной информации, относящейся к одному сотруднику, являющемуся членом нескольких рабочих групп, выливается в весьма нетривиальную задачу). Для решения последней проблемы существует программный продукт *Microsoft SharePoint Portal Server 2003*, обеспечивающий интеграцию узлов *WSPS* в единый *корпоративный портал*.

Работа с общими документами на узле, интеграция с Microsoft Office System

Одной из важнейших функций программного обеспечения для организации групповой работы является упрощение управления общими документами и предоставление средств для совместной работы над документами. В *WSPS* таким средством являются библиотеки документов, которые обеспечивают централизованное хранение документов и доступ к ним как через веб-интерфейс браузера *Internet Explorer*, так и из приложений *Office System* через протокол *WEB DAV 2*.

Для создания библиотеки документов необходимо выполнить следующие действия:

1. В главном меню узла выбрать пункт «Создать»;
2. Выбрать из предложенного списка элементов «Библиотеку документов»;
3. Заполнить имя и описание;
4. Разрешить или запретить отображение имени библиотеки в меню быстрого запуска;
5. Установить переключатель контроля версий документов в положение «Да». Это обеспечит возможность контроля истории изменения документов в процессе работы над ними;
6. Выбрать тип шаблона документа по умолчанию для создаваемой библиотеки как «Документ *Microsoft Office Word*».

Выполнив описанные шаги, создадим несколько библиотек документов: «*Приказы», «Учебные планы», «Методические указания»*. Войдя в одну из созданных библиотек, добавим в нее документ, нажав кнопку «Создать». Набив несколько строк, сохраним документ и закроем *Word*.

Контекстное меню, раскрываемое при щелчке на созданном файле в библиотеке документов, демонстрирует возможные операции над документом (рис. 17).

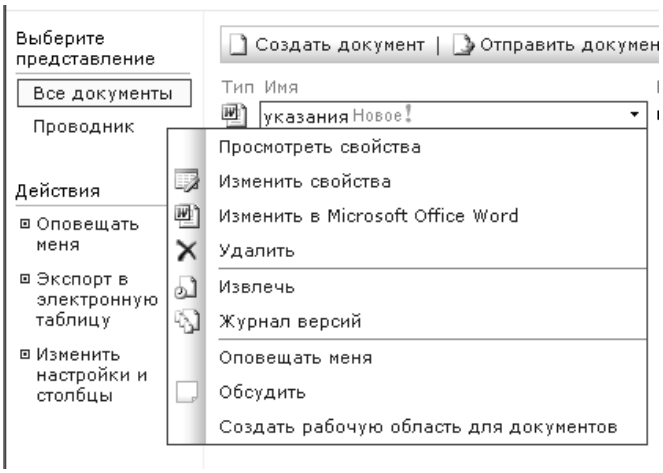

Рис. 17. Контекстное меню документа

При работе над документами в среде *WSPS* используется принцип *check-out/check-in*, т.е. прежде чем начать длительную работу над документом пользователь должен извлечь его из хранилища, тем самым заблокировав возможность редактирования этого документа другими пользователями узла. После завершения работы над документом (при его сохранении) пользователю предлагается вернуть документ, сделав пометки к текущей версии.

Журнал версий используется для контроля истории изменения документа и предоставляет возможность сделать возврат на любую из предыдущих версий (рис. 18).

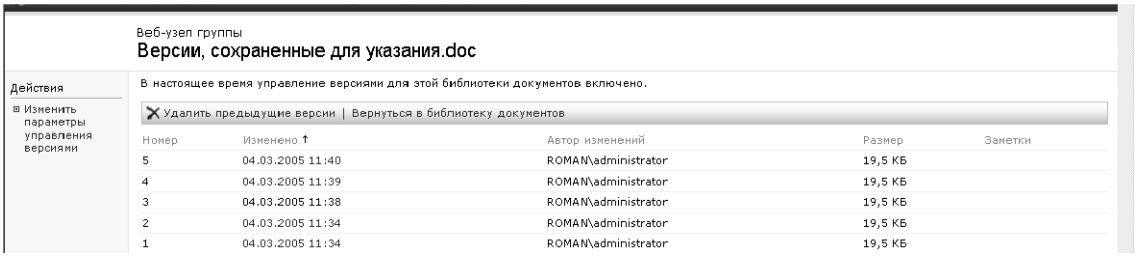

Рис. 18. Журнал версий документа

Информацию о любых изменениях в выбранных документах можно получить, настроив оповещения соответствующим образом. В данной работе этот вопрос не рассматривается, так как настройка оповещения требует дополнительной установки почтового сервера.

Обсуждения дают возможность организовать работу над одним документом нескольким пользователям в режиме онлайн.

При формировании рабочей области для документов создается узел групповой работы над выбранным документом, включающий в себя папку с рабочими документами, список задач, которые необходимо выполнить участниками работы над документом, и другие элементы узла. При этом все действия, выполняемые над документом в рабочей области, не отражаются на базовой копии документа в библиотеке до тех пор, пока не будет осуществлен возврат выбором пункта меню *«Опубликовать в исходном местоположении»*.

Управление задачами и планирование собраний, узлы собраний

Средства групповой работы *WSPS* обеспечивают возможность управления распределением задач между членами рабочей группы. Выбрав пункт *«Списки»* в меню быстрого запуска, можно перейти в список *«Задачи»*, содержащий назначенные исполнителям задания.

Для назначения нового задания необходимо нажать кнопку *«Создать элемент»* в меню списка и заполнить предлагаемые поля. К задаче может быть прикреплен один или несколько файлов (рис. 19).

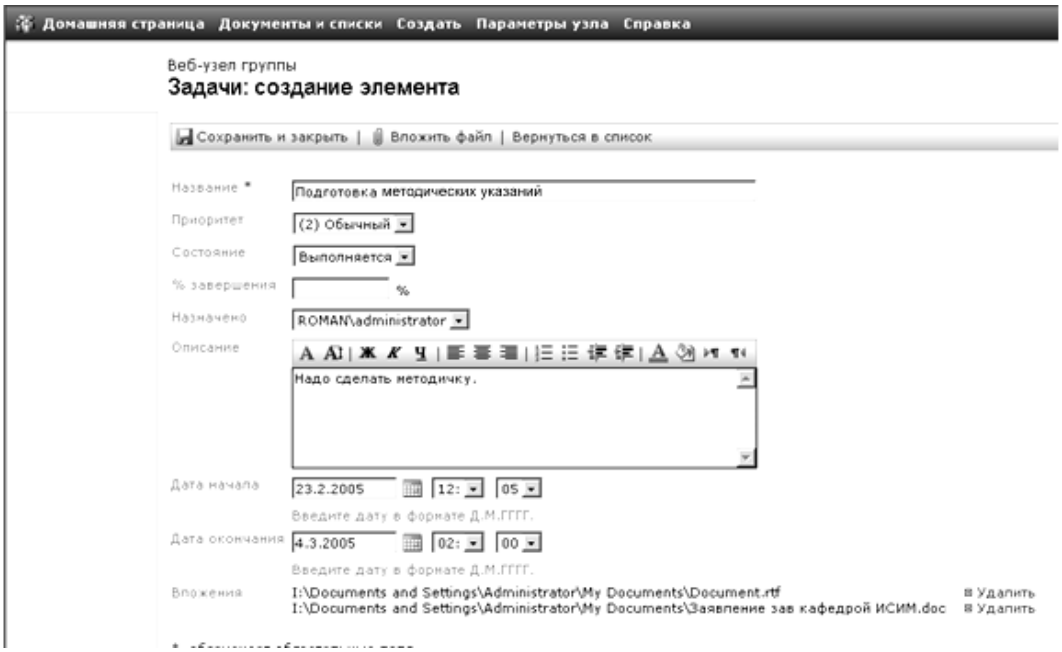

Рис. 19. Назначение новой задачи

При работе со списком задач (и со списками на узле *WSPS* вообще) можно *установить условия фильтрации* выводимых сведений, что суще-

ственно упростит работу со списками, содержащими десятки и сотни записей.

Еще один вид списка является очень важным инструментом при организации коллективной работы над проектами. Список *«События»* можно представить как в виде табличного перечисления элементов, так и в виде календаря с разбивкой по месяцам, неделям и дням (рис. 20).

Рис.20. Представление календаря событий на месяц

Важным моментом в работе с календарем событий является возможность создавать повторяющиеся события – ежедневные, еженедельные и т.д. Кроме этого, специально для крупных событий предусмотрена возможность создания *отдельных узлов*, с помощью которых осуществляется взаимодействие членов рабочей группы при подготовке материалов. Для этого отмечают наличие рабочей области собрания при создании новой записи в списке. Далее будет предложено описать создаваемый узел, указав его название и *URL*, а также выбрать тип шаблона узла из представленного списка.

Так, например, при выборе *шаблона «Решения»* на домашней странице будут собраны несколько специальных списков: «цели», «повестка», «библиотека документов», «задачи», «решения», «участники».

Создание списков произвольного содержания

При работе с разнородной информацией возникает необходимость ее хранения в виде унифицированных списков. При этом может потребоваться связывать между собой списки по каким-то ключевым полям. Службы *WSPS* предоставляют возможность создания настраиваемых списков. В нашем примере создадим два *настраиваемых списка*: *«Группы»* и *«Сту-*

денты» и покажем основные возможности при работе с этим инструментом в среде *WSPS*.

1. Выбрать в главном меню узла пункт «Создать»;
2. Из предложенного списка выбрать вариант «Настраиваемый список»;
3. Задать имя списка – «Группы» – и разрешить вывод названия в меню быстрого запуска, создать список;
4. Находясь в окне списка в меню быстрого запуска, выбрать пункт «Изменить настройки и столбцы»;
5. В разделе «Столбцы» выбрать пункт «Добавить новый столбец»;
6. Ввести название столбца «Информация о группе», выбрать тип столбца «Многострочный текст» и создать столбец.

В результате мы получим пустой список «Группы», состоящий из двух полей «Название» (создается автоматически) и «Информация о группе». Создадим несколько элементов в этом списке и перейдем к созданию списка «Студенты», выполнив шаги 1 – 5 по приведенному выше примеру. Далее выполним следующие действия для настройки списка и связывания данных с информацией из списка «Группы»:

1. Ввести название столбца «Группа», выбрать тип столбца *«Подстановка (данные, уже имеющиеся на этом узле)»;*
2. Отметить требования, чтобы столбец обязательно содержал данные;
3. В выпадающем меню *«Получать данные из»* выбрать пункт *«Группы»;*
4. В меню «Этот столбец» выбрать пункт «Название»;
5. Добавить столбец;
6. Переименовать столбец «Название», выбрав его из списка в разделе *«Столбцы»;*
7. Добавить несколько столбцов, описывающих информацию о студенте (по желанию);
8. Создать несколько записей в созданном списке, внеся в него студентов из разных групп.

В итоге при работе со списком «Студенты» будет отображаться вся внесенная информация. Для настройки отображаемых полей используется механизм настройки представлений списков на узле. С ним можно самостоятельно познакомиться на странице настройки списка, создав укороченное представление (только имя студента и группа) и назначив это пред-

ставление по умолчанию. При этом список будет иметь следующий вид, изображенный на рис. 21.

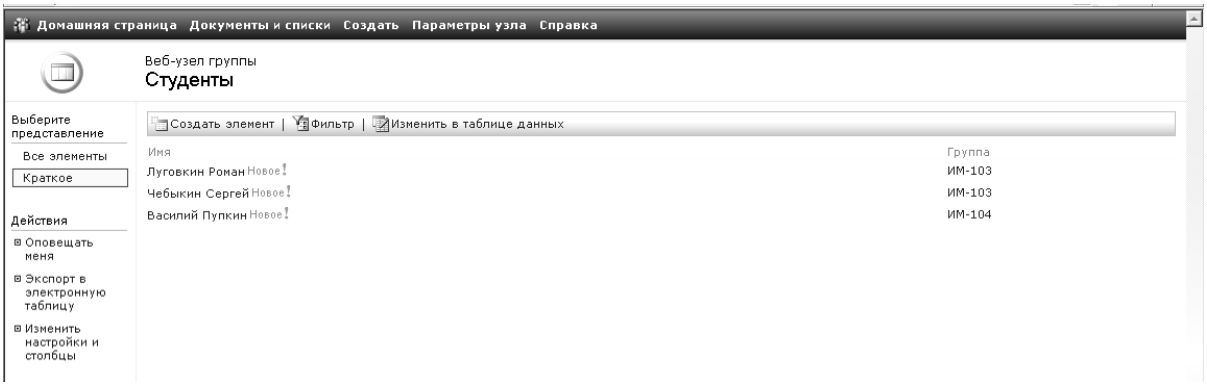

Рис. 21. Краткое представление списка «Студенты»

Кнопка *«Фильтр»* в меню списка позволяет настроить режим фильтрации выводимой информации. Например, можно вывести только студентов заданной группы.

Настройка элементов узла под пользователя, веб-части (web-parts) и компоненты

Компонентная архитектура служб *WSPS* позволяет настроить интерфейс узла под потребности рабочей группы или по желаниям каждого члена рабочей группы. Все отображаемые элементы интерфейса узла являются отдельными *веб-частями (web-parts)*, расположение и внешний вид которых можно настраивать в соответствии с желаниями пользователя.

Для демонстрации возможностей настройки интерфейса воспользуемся рабочей областью для собраний, созданной по ходу выполнения работы.

В правой верхней части окна присутствует кнопка *«Изменить эту рабочую область»* с выпадающим меню, которое иллюстрирует возможности по изменению структуры и внешнего вида страницы узла (рис. 22).

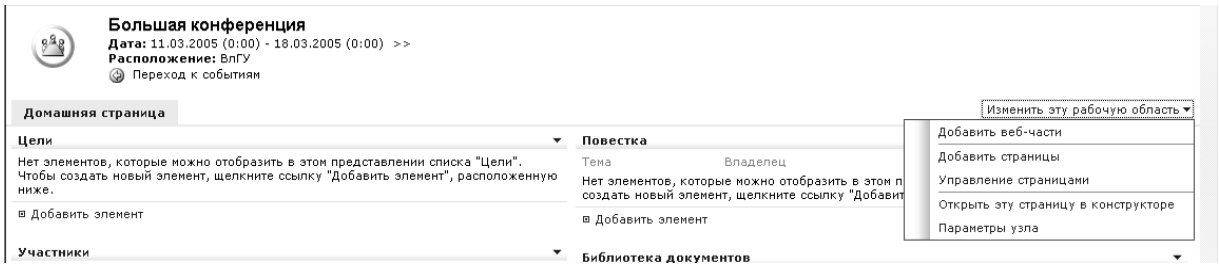

Рис. 22. Меню настройки рабочей области

Выполним следующие действия по настройке рабочей области:

1. В меню выбрать пункт «Изменить эту страницу в конструкторе»;
2. Переместить любую веб-часть из одной зоны в другую, закрыть ненужные веб-части;
3. В меню изменения узла выбрать пункт «Добавить веб-часть», из представленного списка мышкой перетащить пункт «Необходимые предметы» в одну из зон редактируемой страницы;
4. В локальном меню добавленной веб-части выбрать «Изменить общую веб-часть»;
5. В списке «Выбранное представление» выбрать вариант «Мои элементы» и нажать кнопку «Применить», тем самым обеспечить отображения только тех элементов в списке необходимых предметов, которые требуется взять текущему пользователю;
6. В меню изменения узла выбрать пункт «Добавить страницы» и создать страницу «обсуждение календаря»;
7. В открывшемся меню «добавление веб-части» выбрать пункт «Показать все списки» и создать веб-часть «События»;
8. После создания веб-части перейти на «Домашнюю страницу» узла и затем на страницу «Обсуждение календаря», открыв ее в конструкторе;
9. Изменить представление созданной веб-части на представление «календарь», как это было описано выше;
10. Добавить веб-часть «Общие обсуждения» в среднюю зону страницы;
11. Добавить веб-часть «Решения» в правую зону страницы;
12. Переместить мышкой веб-часть «Общие обсуждения» под календарь.

Выполненные действия демонстрируют основные приемы работы с конструктором интерфейса узлов и страниц среды *WSPS*.

Вопросы для самоконтроля

1. Какие методы (уровни) управления можно выделить при рассмотрении места технологии *groupware* в информационной системе компании?
2. Опишите особенности технологии *groupware* и ее концептуальные отличия от технологии *workflow*.
3. Какие функции выполняет технология *Windows SharePoint Services* и в чем особенность программного продукта *Windows SharePoint Portal Server 2003*?
4. Какие основные объекты используются на узлах *SharePoint*?

Практическое занятие № 7
Создание онтологии на языке *OWL*

Цель работы: изучить семантические сети как средство представления знаний и получить навыки описания предметной области в виде семантической сети при помощи языка веб-онтологий *OWL*.

Теоретические сведения

Язык веб-онтологий *OWL* – это язык для определения и представления веб-онтологий. Онтология – термин, заимствованный из философии, который обозначает науку, описывающую формы бытия и отношения между ними.

Веб-онтология может включать описания классов, свойств и экземпляры. Каждой онтологии сопоставлено уникальное пространство имен (как правило, в форме *URL*).

Формальная семантика *OWL* описывает, как получить логические следствия из онтологии, т.е. получить факты, не представленные в онтологии буквально, но следующие из ее семантики. Эти следствия могут быть основаны на одном документе или множестве распределенных документов, которые комбинируются с использованием определенных механизмов *OWL*.

OWL обеспечивает три различных по выразительности диалекта:

- *OWL Lite* – это простейший диалект, предназначенный для построения простой классификационной иерархии с простыми ограничениями. Благодаря этому *OWL Lite* обеспечивает быструю миграцию тезаурусов и других таксономий;
- *OWL DL* базируется на дескриптивной логике – дисциплине, в которой изучен разрешимый фрагмент логики предикатов первого порядка. Благодаря этому возможно автоматически строить иерархию классов и определять ее непротиворечивость;
- *OWL Full* – наиболее выразительный диалект, не поддерживаемый средствами автоматической категоризации.

Экземпляры представляют собой объекты предметной области. В *OWL* для экземпляров не используется предположение об уникальности имен, то есть два разных имени могут относиться к одному и тому же экземпляру.

Класс в *OWL* представляет собой множество экземпляров.

Классы образовывают иерархию обобщения (родительский класс – подкласс), иначе называемую таксономией. Средства автоматической категоризации на основании онтологии проверяют непротиворечивость иерархии классов и достраивают ее. Например, онтология пиццы может содержать следующие классы: *Pizza* для самой пиццы, *PizzaBase* для основы пиццы и *PizzaTopping* для начинки.

В *OWL* существует класс *owl:Thing*, содержащий все объекты. Данный класс является родительским по отношению ко всем остальным именованным классам.

В *OWL* поддерживается множественная классификация, то есть экземпляр может принадлежать сразу нескольким классам. Если никакой экземпляр не принадлежит одновременно более чем одному из классов из некоторого их множества, то данные классы называются непересекающимися.

Свойства представляют собой бинарные отношения между экземплярами. В *OWL* определено три типа свойств: 1) свойства-объекты, связывающие между собой экземпляры; 2) свойства-значения, связывающие экземпляры и значения типов данных, определяемых *XML*-схемой, либо *RDF*-литералы; 3) аннотации, добавляющие метаданные к классам, экземплярам или свойствам. В *OWL* у свойств могут быть подсвойства, то есть свойства, как и классы образуют иерархию. У свойства-объекта может быть обратное ему свойство-объект.

Для свойства-объекта выделяют область определения и область значения. В качестве как того, так и другого может выступать объединение нескольких классов.

Свойство-объект может обладать определенными характеристиками, обогащающими его значение.

Если свойство P – транзитивное, тогда для любого x, y и z: $P(x, y)$ и $P(y, z)$ предполагают $P(x, z)$.

Если свойство P – симметричное, тогда для любого x и y: $P(x, y)$, если $P(y, x)$.

Если свойство P – функциональное, тогда для любого x, y и z: $P(x, y)$ и $P(x, z)$ предполагают $y = z$.

Если свойство P – обратно функциональное, то для всех x, y и z: $P(y, x)$ и $P(z, x)$ предполагают $y = z$.

Например, в онтологии пиццы можно определить транзитивное свойство *hasIngredient* (имеет ингредиент), у которого есть обратное свойство *isIngredientOf* (является ингредиентом), а также два подсвойства – *hasTopping* (имеет начинку) и функциональное свойство *hasBase* (имеет основу). Для двух последних может быть задана область определения (класс *Pizza*) и область значений (классы *PizzaTopping* и *PizzaBase* соответственно).

Множество экземпляров, составляющих класс, задается формально с помощью ограничений трех видов: 1) кванторных ограничений; 2) ограничений кардинальности и 3) ограничений на специфическое значение свойства. Ограничения всегда определяются для того или иного свойства. Таким образом, именно свойства определяют состав классов. По сути, ограничение представляет собой анонимный класс, в него входят те экземпляры, для которых выполняется ограничение. Если для класса определено несколько ограничений, то экземпляр класса должен удовлетворять всем из них, то есть экземпляр должен входить в пересечение всех анонимных классов, соответствующих ограничениям.

Наиболее простыми являются кванторные ограничения, состоящие из трех компонентов: квантора существования или квантора всеобщности, имени свойства и заполнителя. Заполнитель – это анонимный или именованный класс, экземпляры которого выступают в качестве значений свойства.

Например, ограничение $\exists hasBase\ PizzaBase$ описывает все экземпляры, имеющие не менее одной связи с экземплярами класса *PizzaBase* по свойству *hasBase*.

В *OWL* класс может быть описан или определен (соответственно класс называется примитивным или определенным). Примитивный класс содержит только необходимые ограничения, а определенный класс – необходимые и достаточные.

Онтологии, описанные на языке *OWL DL*, могут обрабатываться блоком рассуждений, который выполняет следующие функции:

1. Категоризация онтологии, то есть логический вывод родовидовых отношений между классами онтологии (иначе говоря, построение иерархии классов). Рекомендуется при разработке онтологии использовать только одиночное наследование, оставляя определение мно-

жественного наследования блоку рассуждений. Это позволяет повысить модульность и повторное использование онтологии, облегчить ее поддержку и уменьшить количество человеческих ошибок. Важно отметить, что в общем случае блок рассуждений не порождает подклассы для примитивных классов, обратное, как правило, свидетельствует об ошибке в онтологии;

2. Проверка непротиворечивости онтологии. Если в результате анализа ограничений класса блок рассуждений обнаруживает, что класс не может иметь экземпляров, то такой класс считается противоречивым. При разработке онтологии рекомендуется добавлять в нее противоречивые классы с целью проверки правильности онтологии. Наиболее простой вид такого класса-пробника – класс, наследующий двум непересекающимся классам.

В *OWL* используется открытое допущение (*open world assumption*), которое заключается в следующем: нельзя полагать, что нечто не существует, если это не утверждается явно. Иначе говоря, если истинность чего-либо не утверждается, то это не означает его ложность – считается, что это «знания, не добавленные в базу знаний». Открытое допущение часто приводит к неожиданным результатам при автоматической категоризации.

Открытое допущение имеет одно практическое следствие. Чтобы указать, что некоторый класс может иметь только определенные в ограничениях значения какого-либо свойства, недостаточно использовать ограничения существования. Необходимо определить для данного свойства так называемую аксиому замыкания: ограничение всеобщности, в качестве заполнителя в котором выступает объединение заполнителей из ограничений существования для этого свойства.

Задание

Разработать описание предметной области в виде онтологии, удовлетворяющее следующим требованиям:

1. Онтология должна содержать несколько классов, причем среди них должны встречаться как примитивные, так и определенные классы.
2. Онтология должна содержать несколько свойств-объектов, в том числе вложенных и обратных. Для свойств представить описание различных характеристик (функциональность, транзитивность, симметричность), а также области определения и значения.

3. Онтология должна содержать несколько экземпляров.
4. Описания (определения) классов онтологии должны включать кванторные ограничения и аксиомы замыкания.
5. Онтология должна демонстрировать автоматическую категоризацию и проверку непротиворечивости.

Предварительные требования

Для выполнения работы на компьютере должно быть установлено следующее ПО:

- *JDK* версии 1.2 или выше;
- редактор онтологий *Protégé* 3.1;
- пакет *RacerPro* 1.9.

Порядок выполнения работы

1. Для создания онтологии используется редактор онтологий *Protégé* 3.1, установленный в конфигурации *Basic + OWL*. В качестве блока рассуждений используется *RacerPro* 1.9.
2. Перед созданием собственной онтологии рекомендуется пройти обучающий курс по проектированию онтологии пиццы, который размещается в файле *ProtegeOWLTutorial.pdf*.
3. Для редактирования онтологии требуется создать новый проект (пункт меню *File* / *New Project*). В мастере создания проекта требуется: 1) выбрать тип проекта *OWL Files*; 2) указать пространство имен онтологии по умолчанию (поле *Default Namespace*); 3) выбрать диалект *OWL DL* (группа переключателей *Language Profile*).
4. При открытии проекта в *Protégé* появляются вкладки для редактирования классов (*OWLClasses*), свойств (*Properties*), экземпляров (*Individuals*), форм ввода экземпляров (*Forms*) и метаданных (*Metadata*). В рамках практического задания достаточно использовать первые три.
5. Для определения классов предназначена вкладка *OWLClasses*, общий вид которой приведен на рис. 23. Слева на вкладке отображается иерархия классов (поз. 1), в ее панели инструментов размещаются кнопки для создания подкласса (*Create Subclass*), класса на том же уровне вложенности (*Create Sibling Class*) и удаления класса (*Delete Class*).

6. Для создания класса следует выбрать родительский класс в иерархии классов и нажать кнопку *Create Subclass* либо выбрать класс, который будет соседствовать в иерархии с создаваемым, и нажать кнопку *Create Sibling Class*. Создаваемому классу автоматически присваивается имя, которое следует изменить в поле *Name* (поз. 2 на рис. 23).

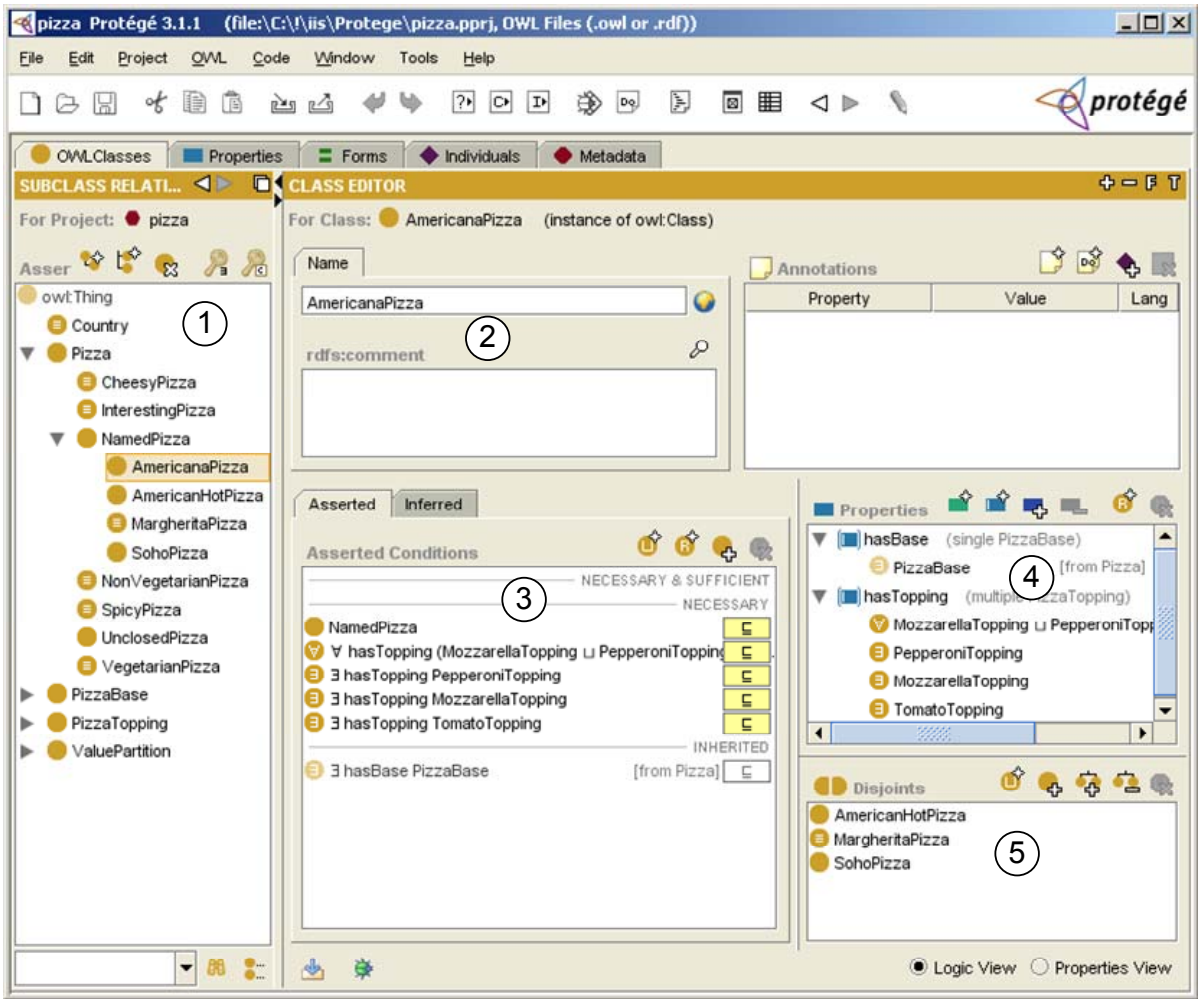

Рис. 23. Редактор классов онтологии

7. Для задания списка непересекающихся классов предназначено окно *Disjoints* (поз. 5 на рис. 23). Чтобы добавить в этот список все классы, имеющие тот же родительский класс и находящиеся на том же уровне вложенности, что и выбранный в иерархии класс, предназначена кнопка *Add all siblings*.

8. Существует альтернативный способ создания иерархии классов с помощью мастера создания множества подклассов (рис. 24), который вызывается пунктом главного меню *Tools* / *Quick OWL* / *Create multiple subclasses*. На первом экране мастера можно указать одинаковый префикс или суффикс для всех создаваемых классов, а в текстовом поле следует указать названия классов по одному на строке. Если установлен флажок *Tab indent to create hierarchy*, то с использованием символа табуляции можно создать иерархию классов (один символ табуляции соответствует одному уровню вложенности). На втором экране мастера можно сделать все классы, создаваемые на одном уровне вложенности, непересекающимися, установив флажок *Make all primitive siblings disjoint*.
9. После создания базовых классов онтологии следует определить свойства онтологии. Для определения свойства, областью определения которого является выбранный класс, можно воспользоваться окном *Properties* на вкладке *OWLClasses* (поз. 4 на рис. 23), но рекомендуется определять свойства и задавать их характеристики на вкладке *Properties*, общий вид которой приведен на рис. 25. Слева на вкладке отображается иерархия свойств (поз. 1), в панели инструментов которой размещаются кнопки для создания свойства-значения (*Create datatype property*), свойства-объекта (*Create object property*) и вложенного свойства-объекта (*Create subproperty*), а также для удаления свойства (*Delete property*). Создаваемому свойству автоматически присваивается имя, которое следует изменить в поле *Name* (поз. 2 на рис. 25).

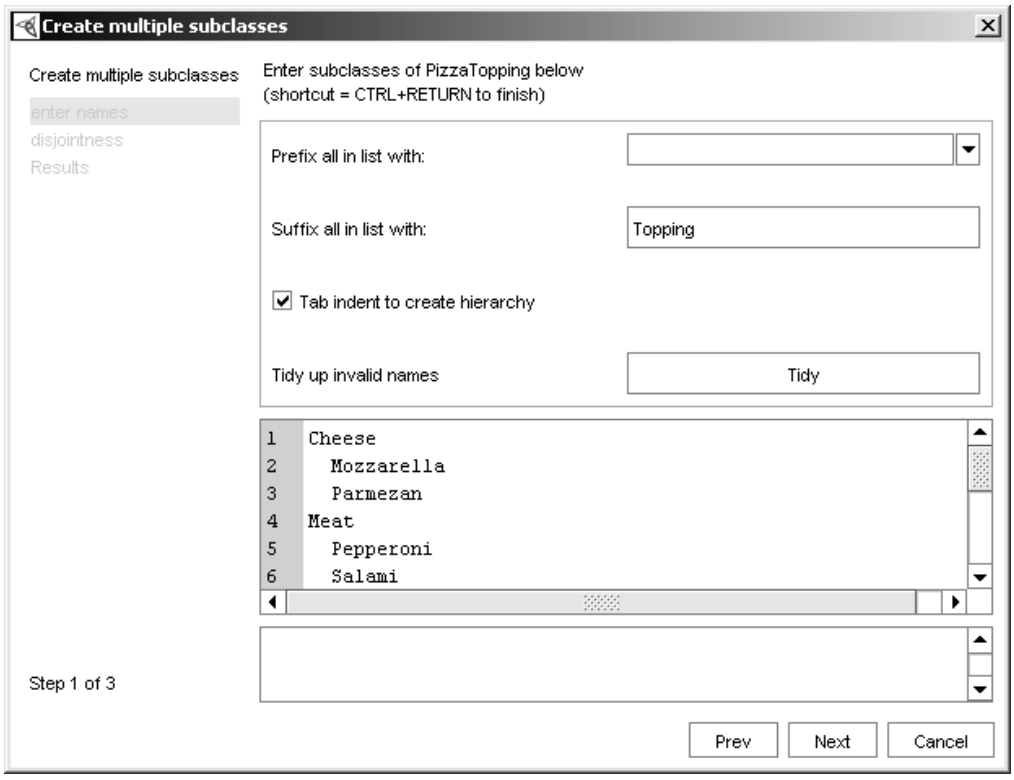

Рис. 24. Мастер создания подклассов

10. Область определения и область значения свойства-объекта задаются в окнах *Domain* и *Range* соответственно (поз. 3 на рис. 25). Для добавления класса онтологии в ту или другую область предназначена кнопка *Add named class*.
11. Характеристики свойства задаются группой флажков (поз. 4 на рис. 25). Свойство, обратное выбранному, определяется в окне *Inverse* (поз. 5 на рис. 25), в котором кнопка *Create new inverse property* позволяет создать новое свойство, а кнопка *Set inverse property* – выбрать уже существующее.
12. После определения свойств онтологии следует вернуться на вкладку *OWLClasses* и завершить описание (определение) классов, задав для них ограничения. Ограничения для выбранного в иерархии класса перечислены в таблице *Asserted* (поз. 3 на рис. 23), где они разделены на необходимые и достаточные (*Necessary & Sufficient*) и просто необходимые (*Necessary*). Ограничение можно создавать в виде выражения или с помощью специального диалога (рис. 26).

13. Для создания ограничения с помощью диалога следует выделить тип ограничения в таблице *Asserted* (строку *Necessary & Sufficient* или просто *Necessary*) и нажать кнопку *Create Restriction* в той же таблице. В диалоге создания ограничения нужно:
    13.1) выбрать вид ограничения в списке *Restriction*. Ограничению существования соответствует вид ограничения *someValuesFrom*, а ограничению всеобщности – *allValuesFrom*;
    13.2) выбрать свойство, на значение которого устанавливается ограничение, в списке *Restricted Property*;
    13.3) указать заполнитель в поле *Filler*. Наиболее простой вид заполнителя – именованный класс, при этом можно не вводить имя класса целиком, а воспользоваться подсказкой, которая появляется при нажатии сочетания клавиш *Ctrl+Space*. При использовании в качестве заполнителя анонимного класса соответствующее ему выражение можно построить с помощью панели ввода выражений, размещающейся под полем *Filler*.
14. Аксиому замыкания для некоторого свойства можно создать вручную, с помощью диалога создания ограничения либо автоматически, выбрав в таблице ограничений *Asserted* одно из ограничений существования и выполнив пункт *Add closure axiom* из его контекстного меню.
15. Перед выполнением автоматической классификации следует уточнить у преподавателя адрес доступа к блоку рассуждений и установить его в поле *Reasoner URL* на вкладке *General* диалога установки предпочтений, который вызывается пунктом главного меню *OWL* / *Preferences*.
16. Для выполнения автоматической категоризации и проверки непротиворечивости предназначен пункт главного меню *OWL* / *Classify taxonomy*. Результаты классификации отображаются: 1) в появляющейся панели *Classification Results* в виде таблицы со столбцами для классов и найденных родительских классов; 2) в окне *Subclass Relationships – Inferred Hierarchy* в виде дерева классов. В последнем новые подклассы выделяются синим цветом шрифта, а противоречивые классы – красной окантовкой обозначения класса.
17. Для работы с экземплярами предназначена вкладка *Individuals*, общий вид которой приведен на рис. 27. Слева на вкладке отображает-

ся иерархия классов (поз. 1), а справа от нее – таблица экземпляров выбранного класса (поз. 2) и таблица типов выбранного экземпляра (поз. 3).

18. Для создания экземпляра нужно в иерархии выбрать класс, к которому он будет принадлежать, и нажать кнопку *Create Instance* в таблице экземпляров класса (поз. 2 на рис. 27). Создаваемому экземпляру автоматически присваивается имя, его можно изменить в поле *Name* (поз. 4 на рис. 27). Кроме того, необходимо указать значения свойств экземпляра (поз. 5 на рис. 27).
19. После создания всех экземпляров следует выполнить их классификацию, для этого предназначен пункт главного меню *OWL* / *Compute inferred types*. Чтобы в результате классификации для экземпляра были определены новые классы, в которые он входит, рекомендуется изначально создавать экземпляры для наиболее общих классов иерархии.

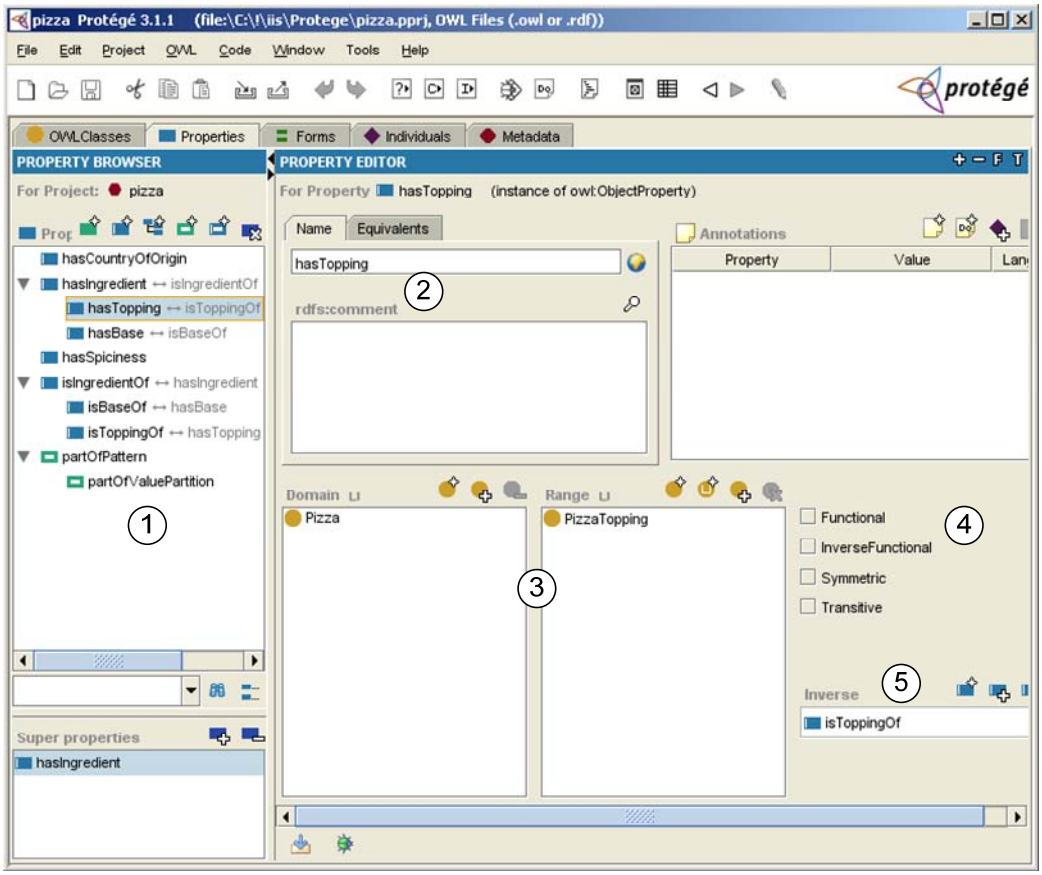

Рис. 25. Редактор свойств онтологии

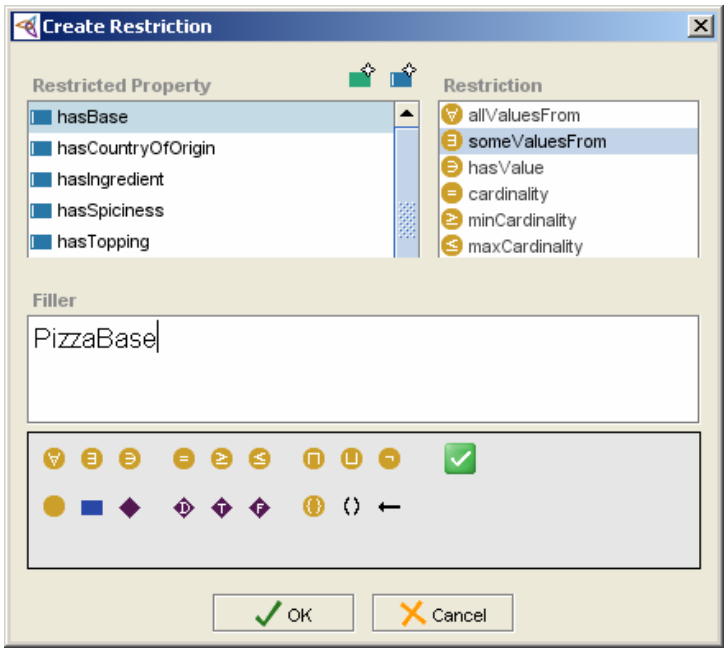

Рис. 26. Диалог создания ограничения

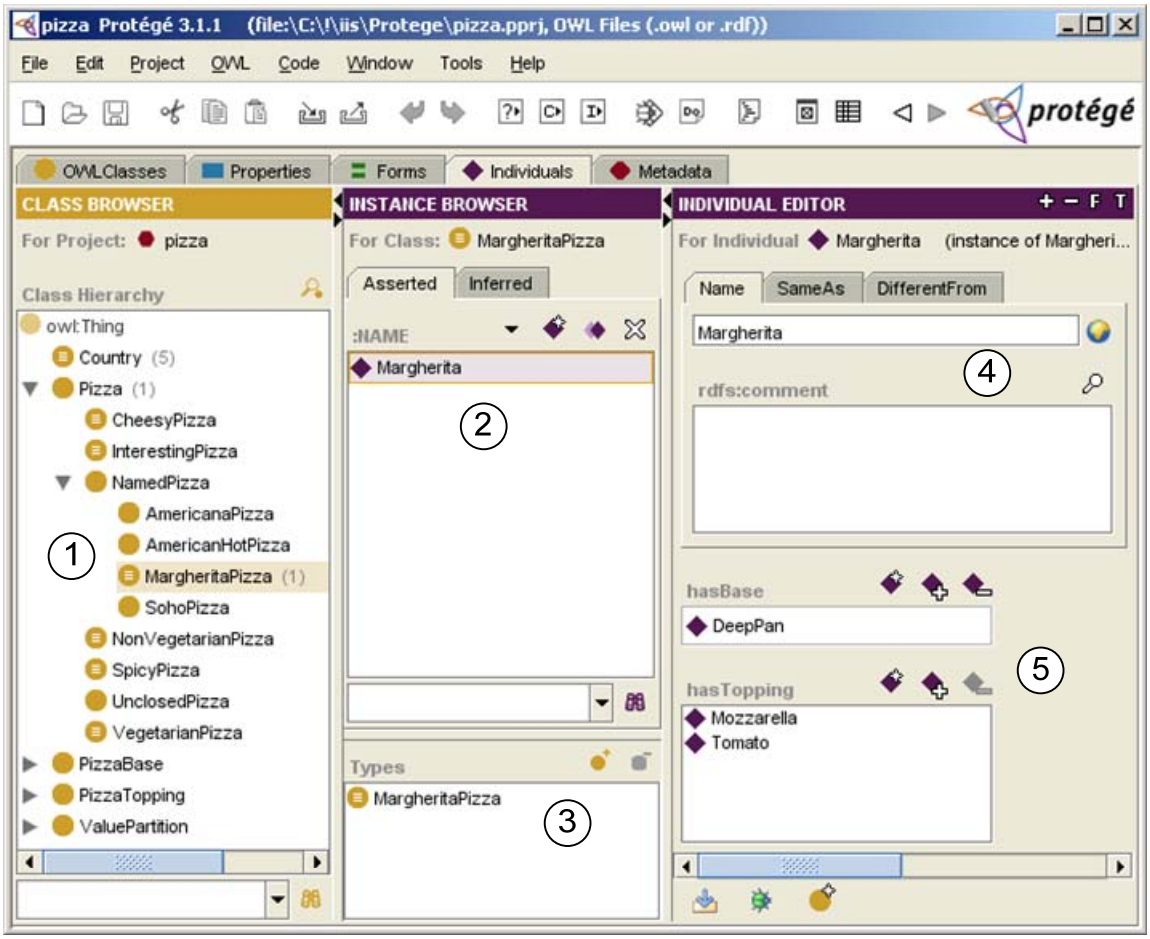

Рис. 27. Редактор экземпляров онтологии

Вопросы для самоконтроля

1. Из каких основных элементов состоит онтология?
2. Какими характеристиками может обладать свойство?
3. В чем состоит отличие между описанием и определением класса?
4. Какие виды ограничений поддерживаются в профиле *OWL* DL?
5. Какие изменения могут быть внесены в онтологию в результате автоматической классификации?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Костров, А. В. Уроки информационного менеджмента. Практикум : учеб. пособие / А. В. Костров, Д. В. Александров. – М. : Финансы и статистика, 2005. – 304 с. – ISBN 5-279-025-73-9.

2. FIPA Agent Management Specification. FIPA TC Agent Management [SC00023J], 12 March, 2002. http://fipa.org

3. Caire, G. JADE programming for beginners. TILAB, 4 December 2003. http://jade.tilab.com

4. Caire, G. JADE Programmer's Guide, TILAB, 10 July 2003. http://jade.tilab.com

5. Caire, G. JADE Administrator's Guide, TILAB, 15 December 2003. http://jade.tilab.com

6. Horridge, M., Knublauch, H., Rector, A., Stevens, R., Wroe, C. A Practical Guide To Building OWL Ontologies Using The Protege-OWL Plugin and CO-ODE Tools. August 27, 2004. http://protege.stanford.edu

7. Specification: Business Process Execution Language for Web Services Version 1.1 http://www.ibm.com/developerworks/library/ws-bpel

ОГЛАВЛЕНИЕ:

ПРЕДИСЛОВИЕ……..3

Практическое занятие № 1. **Основы администрирования СУБД *Oracle***...7

Практическое занятие № 2. **Создание бизнес-процесса в *BPEL*** ……..16

Практическое занятие № 3. **Создание веб-сервиса**……..28

Практическое занятие № 4. **Базовые принципы разработки мультиагентных приложений на платформе *JADE***..…….36

Практическое занятие № 5. **Работа с онтологиями**…….55

Практическое занятие № 6. **Организация групповой работы в среде *Windows Sharepoint Services 2003***……...61

Практическое занятие № 7. **Создание онтологии на языке *OWL***…….76

БИБЛИОГРАФИЧЕСКИЙ СПИСОК …….87

Учебное издание

АЛЕКСАНДРОВ Дмитрий Владимирович
ЖЕБРУН Николай Николаевич
ГРАЧЕВ Иван Викторович

РАСПРЕДЕЛЕННЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ,
ОСНОВАННЫЕ НА ЗНАНИЯХ

Практикум

Подписано в печать 07.07.08.
Формат 60x84/16. Усл. печ. л. 5,11. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.