

Цуканова Н. И.

# Онтологическая модель представления и организации знаний

*Допущено УМО вузов по университетскому политехническому образованию в качестве  
учебного пособия для студентов высших учебных заведений, обучающихся по  
направлению подготовки «Программная инженерия»  
(бакалавриат и магистратура)*

Москва  
Горячая линия – Телеком  
2015

УДК 004.82

ББК 73

Ц85

Р е ц е н з е н т ы : доктор техн. наук, профессор Рязанского государственного радиотехнического университета *И. Ю. Каширин*; доктор техн. наук, профессор Рязанского государственного педагогического университета имени С.А. Есенина *В. Н. Ручкин*

### Цуканова Н. И.

**Ц85**      Онтологическая модель представления и организации знаний. Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2015. – 272 с.: ил.

**ISBN 978-5-9912-0454-5.**

Систематически изложены такие вопросы, как онтология и ее основные компоненты; определение и описание классов; назначение и функции машины вывода (резонера); построение иерархии классов; технология разработки онтологий предметной области; дескрипционные логики, лежащие в основе онтологий; табло-алгоритм для логики ALC; OWL – язык описания онтологий; практическое применение онтологий. Приведены примеры; практические упражнения, выполняемые на компьютере; контрольные вопросы и задания. Следует отметить практическую направленность учебного пособия – большое внимание удалено процессу создания онтологий с использованием CASE-средства редактора Protege 4. Книга будет полезна при изучении курсов «Системы искусственного интеллекта» и «Онтология знаний».

Для студентов вузов, программистов, специалистов в области искусственного интеллекта и баз данных.

**ББК 73**

Адрес издательства в Интернет [WWW.TECHBOOK.RU](http://WWW.TECHBOOK.RU)

Учебное издание

**Цуканова Нина Ивановна**

**Онтологическая модель представления и организации знаний**

*Учебное пособие для вузов*

Компьютерная верстка И. А. Благодаровой

Обложка художника О. В. Карповой

Подписано в печать 26.11.2014. Формат 60×88/16. Уч. изд. л. 17,00. Тираж 500 экз.

ООО «Научно-техническое издательство «Горячая линия – Телеком»

ISBN 978-5-9912-454-5

© Н. И. Цуканова, 2014, 2015

© Издательство «Горячая линия – Телеком», 2015

## **Оглавление**

<b>Введение.....</b>	<b>6</b>
<b>ГЛАВА 1. Онтология предметной области и ее основные компоненты.....</b>	<b>9</b>
Введение.....	9
1.1. Что такое онтология? .....	9
1.2. Разработка онтологии.....	15
Контрольные вопросы .....	39
1.3. Определение и описание классов .....	39
Контрольные вопросы .....	49
1.4. Использование резонера (машины вывода).....	49
Контрольные вопросы .....	66
1.5. Построение иерархии классов.....	66
Контрольные вопросы .....	76
1.6. Характеристики объектов (DataProperties) .....	76
Контрольные вопросы .....	94
Подведем итоги.....	94
Контрольные вопросы .....	103
<b>ГЛАВА 2. Технология разработки онтологии предметной области.....</b>	<b>104</b>
2.1. Методология построения онтологии продукта .....	105
2.2. Проектирование онтологии на основе концептуальной модели предметной области.....	107
Контрольные вопросы .....	123
<b>ГЛАВА 3. Дескрипционная логика .....</b>	<b>125</b>
Введение.....	125
3.1. Общие сведения.....	127
3.2. Синтаксис .....	131
3.3. Синтаксис логики <i>ALC</i> .....	133
3.4. Семантика .....	133
3.5. Семантика логики <i>ALC</i> .....	134
3.6. Связь с логикой предикатов .....	135
3.7. База знаний .....	138
3.8. Аксиомы и <i>TBox</i> .....	138

3.9. Утверждения и <i>ABox</i> .....	141
3.10. Выразительные ДЛ .....	142
<i>Контрольные вопросы</i> .....	144
3.11. Логический анализ .....	144
3.12. Свойства ДЛ .....	146
3.13. Разрешимость логики <i>ALC</i> .....	146
3.14. Понятие разрешающего алгоритма .....	149
3.15. Табло-алгоритм для логики без терминологий .....	151
3.16. Табло-алгоритм для логики <i>ALC</i> с терминологиями .....	153
3.17. Отличие баз знаний от баз данных .....	160
3.18. Связь с языком <i>OWL</i> .....	165
3.19. Машины вывода и редакторы .....	166
3.20. О вычислительной сложности логики <i>ALC</i> .....	167
<i>Контрольные вопросы</i> .....	168
<b>ГЛАВА 4. OWL – язык описания онтологий .....</b>	<b>170</b>
4.1. Основные понятия .....	170
4.2. Конструкции языка <i>OWL</i> .....	171
4.3. Управление онтологиями .....	193
4.4. Связь <i>OWL</i> с другими технологиями .....	199
<i>Контрольные вопросы</i> .....	201
<b>ГЛАВА 5. Практическое применение онтологий .....</b>	<b>203</b>
5.1. Основные области применения онтологий .....	203
5.2. Системы искусственного интеллекта .....	206
5.3. Semantic Web .....	208
5.4. Разработка и управление терминологией .....	212
5.5. Концептуальное моделирование .....	213
5.6. Системы управления знаниями .....	214
5.7. Интеграция разнородных источников данных .....	216
5.8. Спецификация содержимого разнородных источников данных .....	219
5.9. Информационный поиск .....	222
5.10. Семантический поиск .....	237
5.11. Онтологии в электронной коммерции .....	240
5.12. Примеры существующих проектов и систем .....	241
<i>Контрольные вопросы</i> .....	250

<b>Заключение.....</b>	<b>251</b>
<b>Библиографический список .....</b>	<b>253</b>
<b>ПРИЛОЖЕНИЕ П1. Программа «Семья» в формате, основанном на Манчестерском синтаксисе.....</b>	<b>259</b>
<b>ПРИЛОЖЕНИЕ П2. Программа «Семья» в формате, основанном на Функциональном синтаксисе .....</b>	<b>266</b>

*Наука – не разновидность черной магии.  
Порой приходится долго блуждать из  
тупика в тупик, прежде чем будет найдена широкая дорога к научной мысли.*

*Гулд Лоуренс*

## **Введение**

В привычном понимании онтология – это философская категория. В структуре философского знания онтология – это учение о бытии, изучающее фундаментальные принципы бытия, наиболее общие сущности, категории сущего.

В последнее время термин «онтология» стал часто встречаться в литературе по информационным технологиям, где его смысл отличается от принятого в философии. В вузах появились дисциплины, в названии которых присутствует этот термин. В результате возникла потребность понять и разобраться, что означает термин «онтология», и какую роль играют онтологии в информационных системах.

Данное пособие как раз и посвящено целям описания, толкования и рассмотрения областей применения термина «онтология» как модели предметной области в информационных системах.

*Назначение и особенности книги.* Данное пособие представляет собой учебник, включающий в себя разделы теоретического описания онтологий как модели предметной области, практического создания онтологий с использованием редакторов, разделы описания формальных систем, лежащих в основе онтологии, языков описания онтологий и раздел применения онтологий в информационных системах.

В настоящее время в Интернете и в печатных изданиях появилось много публикаций, посвященных этой теме. В большинстве случаев каждая публикация посвящена какому-либо одному вопросу, а литературы, описывающей онтологию, начиная с базовых понятий до практических реализаций мало в силу творческого (пока далекого от стандарта) характера исследований в этой области.

В данном пособии сделана попытка соединить большую часть аспектов, связанных с понятием онтологии: определение, формализмы, языки, технологию разработки, практическое создание с помощью редакторов и области применения онтологий.

Учебное пособие основано на лекционном курсе, который автор в последние годы читает в Рязанском государственном

радиотехническом университете студентам и магистрантам, обучающимся по направлению: «Программная инженерия».

*Учебник преследует следующие цели:*

1. Познакомить читателя с максимально широким кругом понятий, связанных с онтологией как моделью предметной области. Тем самым, сформировать у студента терминологический запас, необходимый для самостоятельного изучения специальной математической и программистской литературы.

2. Предоставить читателю варианты технологий разработки онтологий и привести примеры их использования, тем самым вооружив читателя правилами проектирования онтологий.

3. Показать возможность использования редакторов (CASE-средств) при создании онтологий. Выполнение упражнений позволит студенту овладеть базовыми понятиями и навыками работы в редакторе в процессе создания онтологий.

4. Познакомить читателя с формальными системами и процедурами вывода новых знаний, позволяющими на языке логики описать онтологию предметной области, выявить противоречия в этом описании и построить иерархию концептов предметной области.

5. С помощью примеров практического использования в информационных системах пояснить назначение онтологий.

6. Познакомить читателя с описанием онтологий на языках Semantic Web.

*Структура книги.* Учебное пособие содержит 5 глав. Первая глава носит практический характер, она посвящена процессу создания онтологии Пиццы в редакторе Protege 4. Но целью этой главы в большей степени является определение понятия онтология и знакомство со всеми базовыми терминами, связанными с онтологией. На примерах легче понять сложные категории.

Во второй главе рассматриваются два варианта технологии разработки онтологии предметной области. Очень важно иметь правила или рекомендации, следуя которым любой инженер по знаниям сможет разработать онтологию по заданной предметной области. Такие правила полезны студентам при выполнении лабораторных работ.

Третья глава посвящена дескрипционным логикам – формализмам, лежащим в основе онтологий. Дескрипционные логики [45, 62] – это модели представления знаний в онтологиях. С ними

тесно связана и процедура вывода новых знаний. В основе ее лежит один из базовых алгоритмов: либо алгоритм, основанный на принципе резолюции [27], либо табло-алгоритм [55, 60]. Если алгоритму, основанному на принципе резолюции, посвящено много работ на русском языке, то найти литературу на русском языке о табло-алгоритме сложно, хотя он в настоящее время является основным для большинства резонеров – машин вывода. В пособии суть табло-алгоритма раскрывается на примерах. Приводится без доказательства ряд теорем, связанных с алгоритмическими проблемами в онтологии.

Четвертая глава посвящена языкам описания онтологий, ставшим стандартами в области Semantic Web – это OWL, RDF, XML [41, 46, 48, 50]. Рассматривается простой пример описания семьи, его разработка ведется в редакторе Protege 4, а затем описывается на языке OWL.

В пятой главе сначала приводится перечень различных областей, где может быть использована онтология предметной области. Этот перечень основан на обзоре публикаций последних лет в области онтологий. Затем рассматриваются более подробно примеры практического использования онтологий и анализируется целесообразность применения онтологии в том или ином случае.

В конце книги приводится обширный библиографический список публикаций по теме онтология в информационных системах: на русском языке, на иностранных языках и в электронных ресурсах. Читатель в этом списке может найти интересующую его тему.

По ходу изложения материала в конце большинства разделов приведены контрольные вопросы и контрольные задания. Поиск ответов на эти вопросы и выполнение заданий поможет студентам усвоить учебный материал.

*Благодарности.* Автор выражает благодарность рецензентам Каширину И.Ю., Ручкину В.Н., Пылькину А.Н., которые помогли выявить недочеты в пособии, дали хорошие советы по улучшению качества изложения материала. Особую благодарность хочется выразить коллеге Благодаровой И.А. за неоценимую помощь в оформлении рукописи, а также поблагодарить студентов, прослушавших этот курс, за выявление ошибок в тексте.

*Человек должен непоколебимо верить, что непостижимое постижимо, иначе он ничего не сможет исследовать.*

*Иоганн Вольфганг Гете*

## Глава 1

# ОНТОЛОГИЯ ПРЕДМЕТНОЙ ОБЛАСТИ

## Введение

Данный раздел знакомит читателя с понятием онтологии в информационных технологиях на примере разработки простой онтологии Пиццы. В качестве инструмента (CASE-средства) разработки используется редактор онтологий Protege 4. Сначала вводится понятие онтологии предметной области, затем рассматривается процесс создания онтологии, описание свойств предметной области на языке дескриптивной логики и использование резонера (машины вывода) для проверки согласованности онтологии и автоматического вычисления иерархии классов. Материал данного раздела основан на следующих источниках: Matthew Horridge et al. 2011. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools, Edition 1.3, published by the University of Manchester, 24 Mar 2011, 108 pp., а также [7-8].

### 1.1. Что такое онтология?

Онтологии используются для описания знаний о некоторой предметной области. Онтология описывает понятия предметной области, а также отношения, которые имеются между этими понятиями.

Онтология, как модель, выражает определенный взгляд (разработчика) на некоторую предметную область и формально может быть описана следующим набором множеств [7]:

$$O = \langle X, R, \Phi \rangle,$$

где:  $X$  – конечное множество концептов (понятий, терминов) предметной области, которую представляет онтология  $O$ . Например, для предметной области «Институт» такими понятиями являются *студент, группа, факультет, преподаватель, дисциплина и т.д.*;

$R$  – конечное множество отношений между концептами (понятиями, терминами) заданной предметной области. Например, *студент\_группа, группа\_факультет, студент\_факультет* и т.д.

Здесь имя отношения формируется из имен связываемых этим отношением сущностей и знака подчеркивания \_;

Ф – конечное множество функций интерпретации, заданных на концептах и/или отношениях онтологии О. Роль функции интерпретации может играть словесное пояснение термина (аннотация), формула для вычисления значения термина, алгоритмическое описание, а также определение в виде логической формулы *Отец* ≡ *Мужчина and родитель\_ребенка some Личность*.

Из всего множества отношений в онтологии выделяется специальный класс – простая таксономия:  $O = T^o = \langle X, \{is\_a\}, \{\}\rangle$ .

Под таксономической структурой понимается иерархическая система понятий, связанных между собой отношением *is\_a* («быть элементом класса» или «быть подклассом класса»). Это отношение (*is\_a*) позволяет организовать структуру понятий онтологии в виде дерева. Отношение *is\_a* имеет фиксированную заранее семантику. Предложение «Элемент А является подклассом класса В», описывается простой логической формулой (импликацией):  $A \rightarrow B$  – «*Если A, то B*». Получаем формальную таксономию.

Таким образом, из определения онтологии следует, что основными ее компонентами являются: *Классы (Концепты) или понятия, Индивидуальности (экземпляры), Отношения, Функции, Аксиомы*.

В целом потребность в разработке онтологий объясняется следующими причинами [7]:

- совместное использование людьми или программными агентами [59] общего понимания структуры информации;

- разработка и управление терминологией; язык описания онтологий (OWL) использовался для поддержки огромных терминологических словарей с сотней тысяч терминов и сложной иерархией [64];

- возможность повторного использования знаний в предметной области [61];

- получение надежного семантического базиса в определении содержания;

- отделение знаний в предметной области от оперативных знаний;

- получение логической теории, которая состоит из словаря и набора утверждений на некотором языке логики, что позволяет на

основе этой теории получать вывод новых знаний, явно не заложенных в онтологии [62, 65];

– возможность использования онтологий для поддержки функционирования и роста нового вида цифровых библиотек, реализованных как распределенные интеллектуальные системы.

Процесс создания онтологий включает:

- определение классов в онтологии;
- организация классов в некоторую иерархию (подкласс → класс);
  - определение отношений (связей) между классами, между элементами классов;
  - определение свойств (характеристик, атрибутов) элементов класса;
  - определение экземпляров классов и задание значений их свойств.

В настоящее время для создания и поддержки онтологий существует множество редакторов. В данном учебном пособии для разработки онтологий использовался редактор *Protege 4*.

Рассмотрим его основные достоинства:

- свободно распространяемый программный продукт, что позволяет быстро установить его на любом компьютере;
- дескриптивная логика – тот формализм, на котором основан *Protege 4*;
  - для описания предметной области используются одноместные (*Классы*) и двухместные (*Свойства объектов*) предикаты;
  - синтаксис (*Манчестерский синтаксис*) языка логических формул довольно прост для понимания [67];
- поддерживает импорт и экспорт онтологий в OWL. *Protégé* даже позволяет работать в OWL, т.е. создавать, редактировать, удалять онтологии в формате OWL.

В качестве примера рассмотрим простую и понятную для студентов вуза предметную область, где основными концептами являются понятия «Группа», «Студент», «Факультет» и т.д. На рис. 1.1 приведена онтология предметной области «Учебная деятельность преподавателя». Для этого примера приведены логические формулы, описывающие классы, и формулы, создающие новые отношения.

#### Описание классов

*Студент\_получит\_степень* → *Бакалавр or Магистр or Специалист*

Лекции\_по\_ФЛП → тема\_дисциплина **value** Функциональное\_и\_логическое\_программирование

ФВТ\_студент → студент\_факультет **value** ФВТ

### Описание отношений

В виде цепочки отношений:

лаб\_раб\_дисциплина <- лаб\_раб\_тема о тема\_дисциплина

студент\_дисциплина <- студент\_группа о группа\_дисциплина

студент\_факультет <- студент\_группа о группа\_факультет.

В виде инверсного отношения:

специальность\_группа ≡ **inverse**(группа\_специальность).

Онтология дисциплины может использоваться в учебном процессе как справочник, а также для контроля знаний.

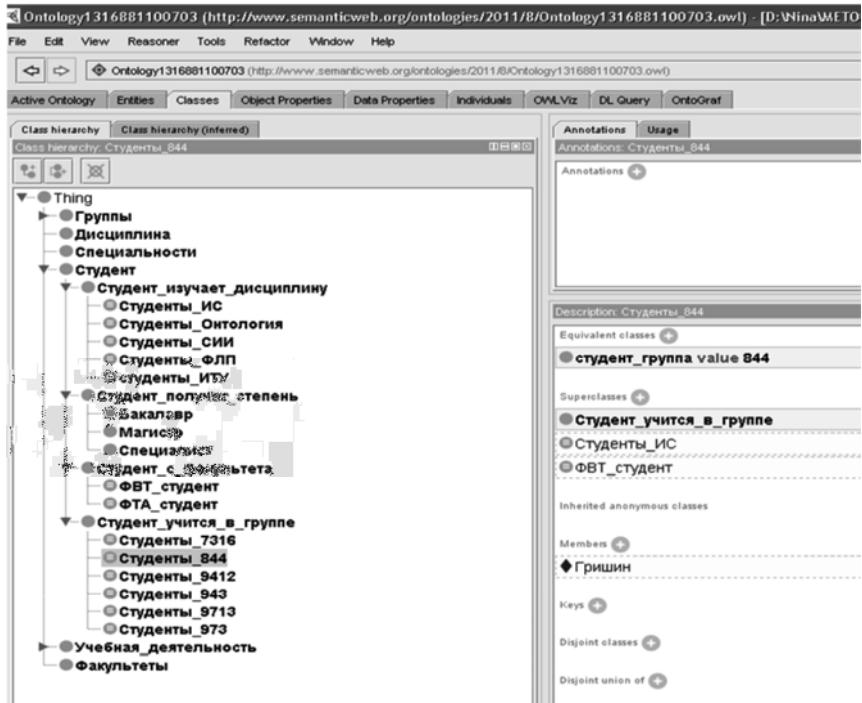


Рис. 1.1. Онтология предметной области «Учебная деятельность преподавателя»

### ЯЗЫКИ ОПИСАНИЯ ОНТОЛОГИЙ

Существуют различные языки описания онтологий, имеющие разные возможности. В последние годы приобрел популярность и

стал стандартом язык OWL (*Web Ontology Language* стандарт *World Wide Web Consortium* (W3C)).

Язык Web-онтологии (OWL) является языком Semantic Web [3], созданным для представления онтологий. Под онтологией будем понимать информацию о группировке отдельных индивидов, которые вместе определяют некоторую предметную область (домен). OWL может представлять информацию о классах индивидов и их свойствах. OWL [41, 63, 69] – это логический язык, где каждая конструкция имеет строго определенный смысл. OWL группирует информацию в онтологии, которые представлены в виде документов. Эти документы могут храниться и передаваться в глобальной сети точно так же, как передаются любые другие данные или информация. Эффективная обработка документов может осуществляться посредством инструментария, позволяющего извлечь информацию, скрытую внутри онтологии.

OWL имеет богатый набор логических операторов, например, пересечение, объединение и отрицание. Благодаря этим операциям сложные понятия определяются через простые концепты. Кроме того, логическая модель позволяет использовать машину вывода (резонер или мыслитель), способную проверить, все ли из утверждений и определений в онтологии взаимно согласованы, а также распознать, какие понятия подходят под какие определения. Автоматические рассуждения, выполняемые резонером, помогают поддерживать иерархию в непротиворечивом состоянии. Это особенно полезно при работе со случаями, когда классы могут иметь более чем одного из родителей.

## **Компоненты онтологии**

Онтология состоит из индивидуальностей, свойств и классов.

### **Индивидуальности или экземпляры классов**

Индивидуальности представляют собой объекты интересующей исследователя области. OWL не использует понятие уникального имени объекта. Это означает, что различные имена могут указывать на один и тот же объект. Так, например, «Королева Елизавета», «Королева» и «Елизавета Виндзор». Возможно, все они относятся к одной и той же особе. В OWL надо обязательно заявить, индивидуальности идентичны или отличны друг от друга. Индивидуальности рассматриваются как экземпляры классов.

На рис. 1.2 показано представление индивидуальностей в некоторой области в виде ромбиков или вершин.



**Рис. 1.2.** Представление индивидуальностей и свойств. Мэтью, Англия, Джемма – индивидуальности, *живетВ*, *имеетБратаИлиСестру* – свойства (отношения)

### **Свойства – бинарные отношения**

Свойство – это бинарное отношение (связь) между двумя индивидуальностями. Так, например, свойство «*Иметь брата или сестру*» связывает личность *Мэтью* с личностью *Джеммой*, свойство «*Иметь ребенка*» может связывать индивидуальность *Петер* с индивидуальностью *Мэтью*. Свойства могут быть инверсными. Например, фраза «сущность **имеет собственника**» является инверсной по смыслу фразе «некто является **собственником**» некоторой сущности. Рис. 1.2 показывает, как свойства (отношения) связывают индивидуальности между собой. В некоторых формализмах свойства называются атрибутами или реквизитами.

### **Классы**

Классы интерпретируются как множества, которые содержат индивидуальности. Сами классы подразделяются на подклассы, а те, в свою очередь, делятся на еще более мелкие классы.

Классы отображаются в виде кругов или овалов (рис. 1.3), как множества в диаграммах Венна. Концепт и класс при описании онтологий рассматриваются как синонимы.

Описания классов строятся в виде условий (утверждений), которым должны удовлетворять индивидуальности для того, чтобы быть членами класса. Как формулировать эти описания, будет объяснено в следующих разделах пособия. Индивидуальности могут принадлежать более чем к одному классу.

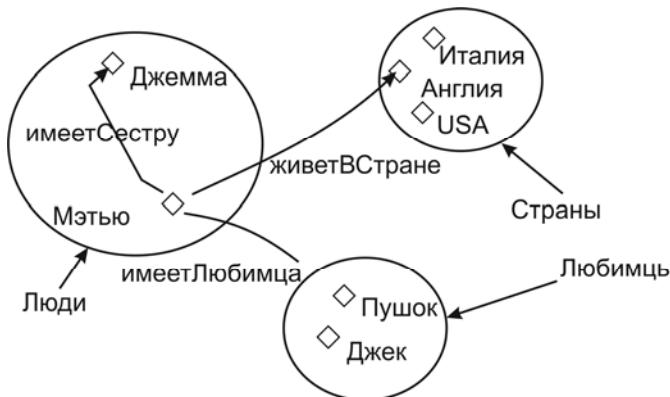


Рис. 1.3. Представление классов, содержащих индивидуальности

## 1.2. Разработка онтологии

Разработку онтологии рассмотрим на примере того, как создать онтологию *Пиццы*. Такая онтология может быть полезна различным интернет-магазинам, кафе, сайтам по кулинарии, заинтересованным в рекламе и распространении информации о пицце. Кроме того, эта предметная область выбрана потому, что в ней много полезных примеров.

**Задание 1:** создайте новую онтологию OWL.

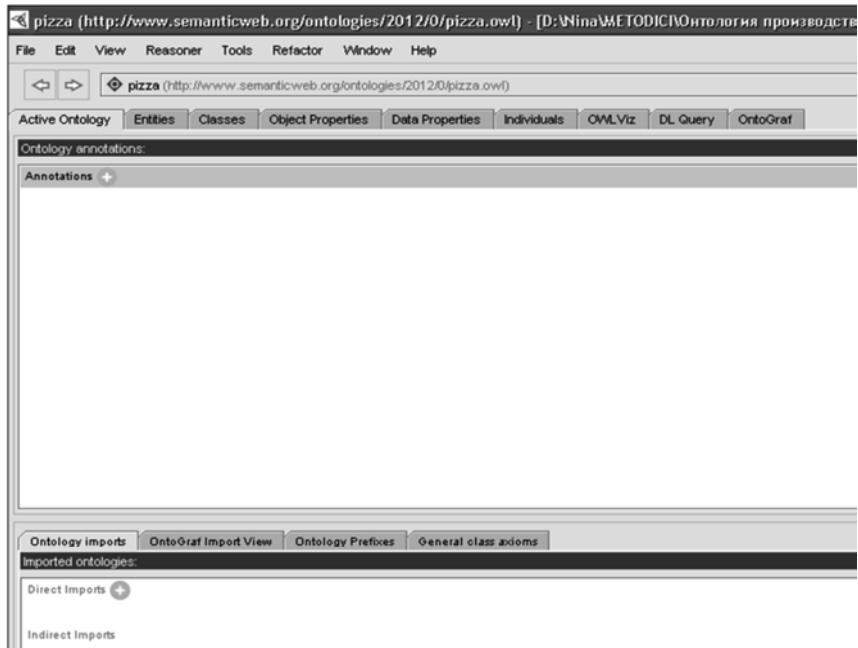
1. Запустите редактор Prot'eg'e 4.
2. В главном меню выберите пункт «File/New» «Создать новую OWL онтологию». В результате появится окно, представленное на рис. 1.4.

3. На вкладке Active Ontology необходимо задать уникальный идентификатор онтологии. Каждая онтология именуется Уникальным идентификатором ресурса (URI) или (IRI – интернациональный идентификатор ресурса). Замените заданный по умолчанию идентификатор с <http://www.semanticweb.org/ontologies/2013/10/un-titled-ontology-119> на <http://www.semanticweb.org/ontologies/2013/10/Pizza> и нажмите «Далее(Continue)».

4. Далее можно сохранить файл онтологии в нужном формате, указав его месторасположение на жестком диске и формат, в котором будет сохраняться онтология (File/Save As...).

Рабочее окно разделено на две части, каждая часть имеет несколько вкладок. На вкладке «Активная онтология (Active

*Ontology)*» можно увидеть и, если нужно, изменить характеристики онтологии. Например, можно изменить URI, добавить аннотацию в виде комментария, настроить пространства имен и импорта.



**Рис. 1.4.** Главное окно редактора *Protege 4*

**Задание 2:** добавьте комментарий к онтологии.

1. Убедитесь, что выбрана вкладка «Активная онтология (Active ontology)».

2. В рабочей части «*Ontology annotations*» нажмите кнопку «Добавить», значок (+) рядом с *Annotations*. Появится окно редактирования (рис. 1.5). Выберите «comment» из списка встроенных типов аннотаций и введите свой комментарий в текстовом поле в правой панели.

3. Введите «*Онтология пиццы описывает классификацию пицц в зависимости от начинки*» и нажмите кнопку ОК, чтобы присвоить комментарий аннотации.

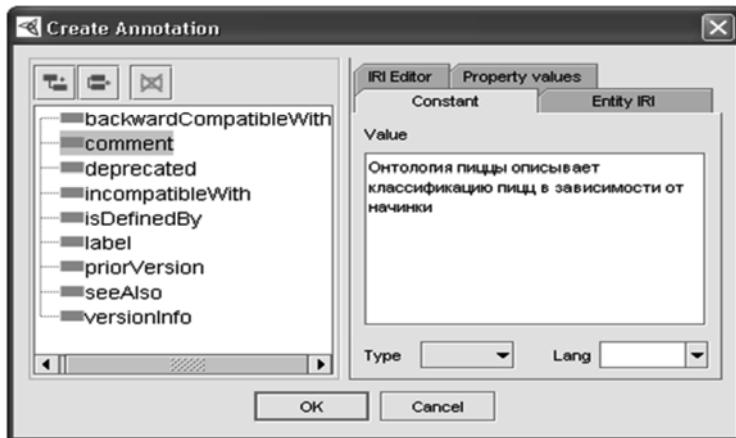


Рис. 1.5. Диалоговое окно «Создание аннотации»

Вкладка «Активная онтология» должна иметь вид, показанный на рис. 1.6.

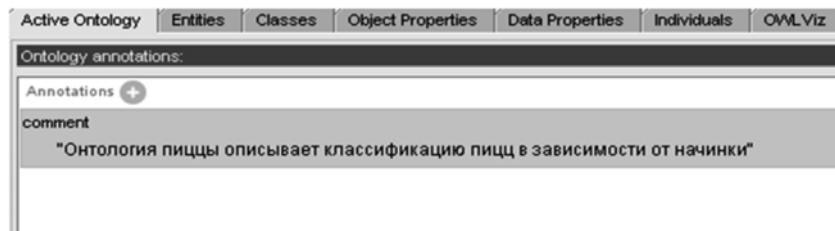


Рис. 1.6. Аннотация онтологии

### Поименованные классы

Как упоминалось ранее, онтология содержит классы. Классы являются основными строительными блоками онтологии. В *Protégé 4* редактирование классов осуществляется на вкладке «Классы» (Classes), которая показана на рис. 1.7.

Пустая онтология содержит один класс, называемый *Thing*. Как упоминалось ранее, классы интерпретируются как множества индивидов (или групп объектов). Класс *Thing* представляет универсальное множество, содержащее все индивидуальности. Из-за этого классы являются подклассами *Thing*. *Thing* является частью словаря OWL, который описывается онтологией, находящейся на <http://www.w3.org/2002/07/owl/\#>.

Давайте добавим несколько классов в онтологию для того, чтобы определить, что является пиццей.

**Задание 3:** создайте классы *Пицца*, *Пицца\_Начинка* и *Пицца\_Основа*

1. Убедитесь, что выбрана вкладка «Классы», выделен класс «*Thing*».

2. Нажмите «Добавить» – значок (⊕), как показано на рис. 1.7. Эта кнопка создает новый класс – подкласс выбранного класса (в данном случае мы хотим создать подкласс *Thing*).

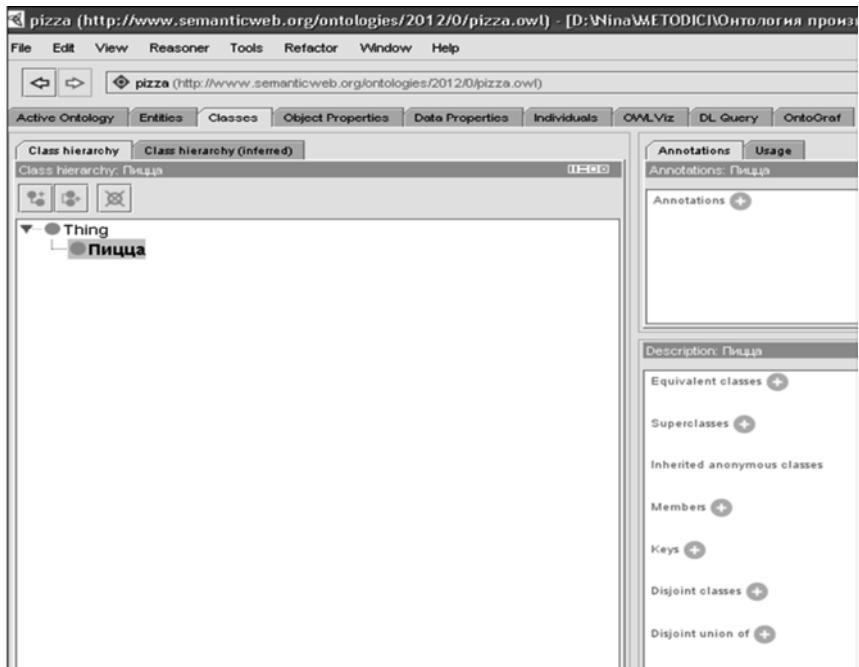


Рис. 1.7. Вкладка «Классы»

3. Появится диалоговое окно для ввода имени класса. Введите «*Пицца*» (как показано на рис. 1.8) и нажмите ОК.

4. Повторите предыдущие шаги, чтобы добавить классы *Пицца\_Начинка*, а также *Пицца\_Основа*, убедившись, что класс *Thing* выбран до нажатия кнопки «Добавить», так что классы создаются как подклассы *Thing*. Иерархия классов должна иметь вид, как показано на рис. 1.9.

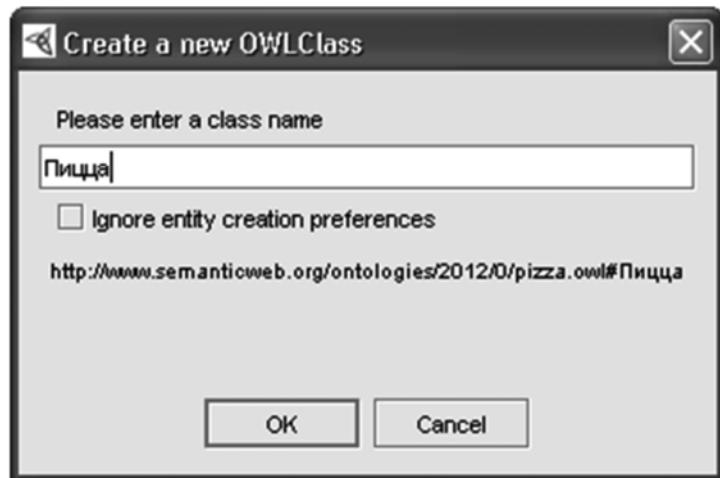


Рис. 1.8. Создание нового класса

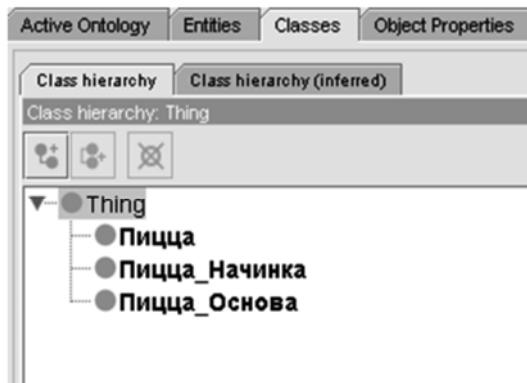


Рис. 1.9. Иерархия классов

После создания Пиццы, вместо того, чтобы повторно выбрать Thing и использовать кнопку «Создание подкласса», можно использовать кнопку «Добавить класс – брат» (см. рис. 1.7). Если выделить класс Пицца и использовать кнопку «Добавить класс-брать», а затем сначала ввести имя «Пицца\_Основа», потом – имя «Пицца\_Начинка», то получим два новых подкласса класса Thing (см. рис. 1.9).

Иерархию классов также называют таксономией. Таксономия – это способ разбиения концептов на группы, подгруппы по тем или

иным критериям (т.е. классификация объектов). Имена классов должны начинаться с буквы, не должны содержать пробелов, и каждое новое слово в имени должно начинаться с большой буквы или отделяться от предыдущего знаком подчеркивания. Выбранное вами правило должно применяться всюду последовательно.

### **Непересекающиеся классы**

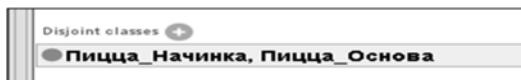
После добавления классов *Пицца*, *Пицца\_Начинка* и *Пицца\_Основа* к онтологии нужно сказать, что эти классы не пересекаются, так что объект не может быть экземпляром больше, чем одного из них.

Чтобы указать классы, которые не пересекаются с выбранным классом, нажмите кнопку (+) в секции «*Disjoint With*», которая расположена справа в нижней части окна «*Описание класса*» (см. рис. 1.7).

**Задание 4:** объявитте классы *Пицца*, *Пицца\_Начинка* и *Пицца\_Основа* не пересекающимися друг с другом.

1. Выберите класс *Пицца* в иерархии классов.

2. Нажмите кнопку (+) (рис. 1.10) в секции «*Disjoint With (Непересекающиеся классы)*» в нижней части окна «*Описание класса*». Это вызовет диалоговое окно, где вы можете выбрать несколько классов и объявить их не пересекающимися с классом *Пицца*. В результате *Пицца\_Основа* и *Пицца\_Начинка* (родной брат класса *Пицца*) не будут пересекаться с классом *Пицца*.



**Рис. 1.10.** Объявление классов, не пересекающихся с классом *Пицца*

### **Использование мастера «Создание иерархии классов»**

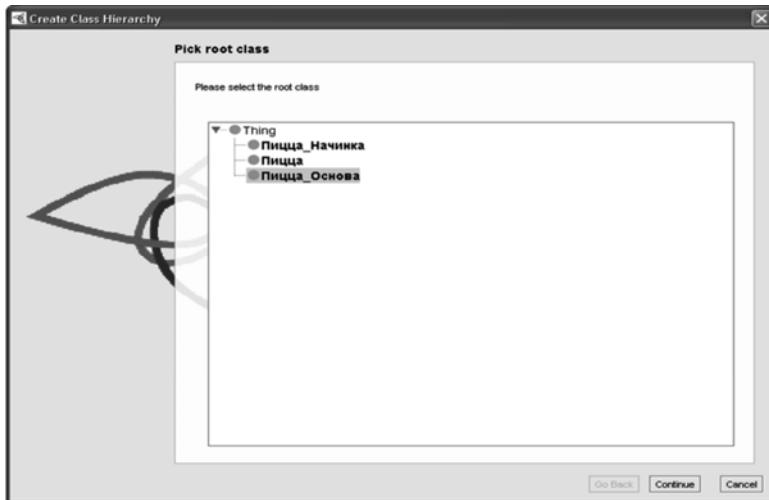
В этом разделе мы будем использовать инструмент «*Создание иерархии классов*», чтобы добавить некоторые подклассы класса *Пицца\_Основа*.

**Задание 5:** воспользуйтесь мастером «*Создание иерархии классов*» для создания классов *Тонкая\_и\_хрустящая* и *Глубокий\_противень* в качестве подклассов *Пицца\_Основа*.

1. Выберите класс *Пицца\_Основа* в иерархии классов.

2. Из меню *Сервис(Tools)* в строке меню *Prot'eg'e* выберите «*Создать Иерархию классов (Create Class Hierarchy)*».

3. Появится окно, представленное на рис. 1.11. Так как предварительно был выбран класс *Пицца\_Основа*, то далее будут создаваться его подклассы. Если ранее класс не был выбран, то в данном окне можно выделить его в дереве концептов.

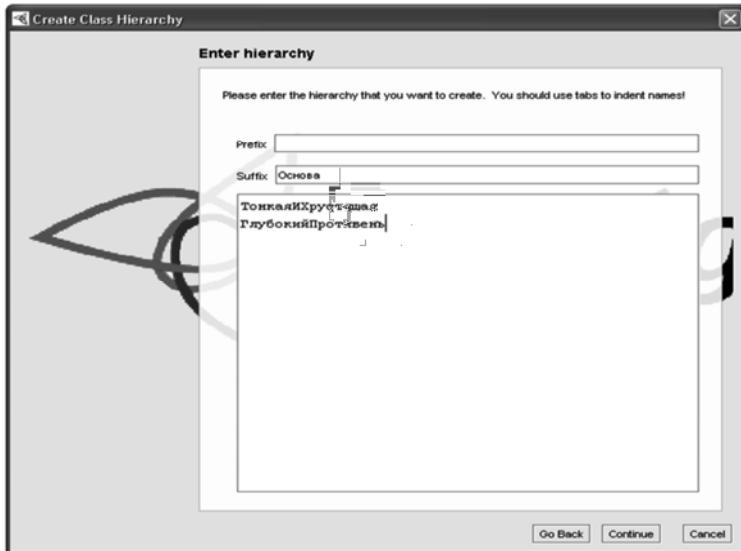


**Рис. 1.11.** Создание иерархии классов с помощью мастера

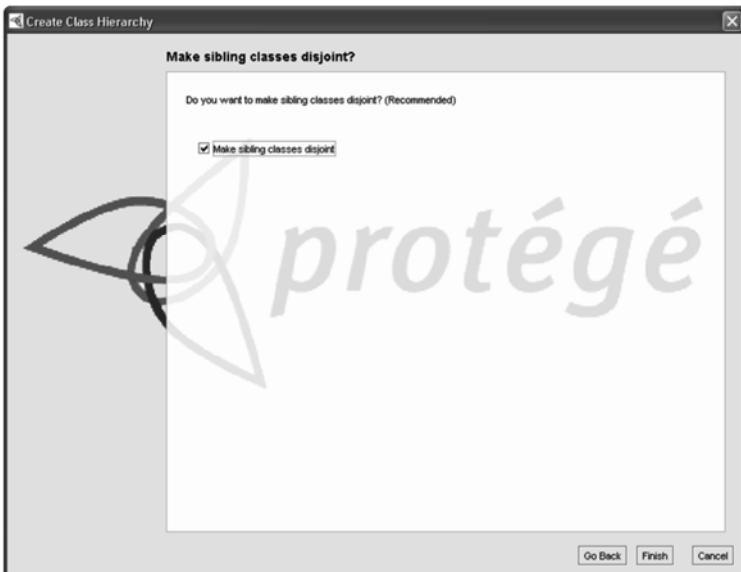
4. После нажатия кнопки «Далее (Continue)» мастера получим изображение, показанное на рис. 1.12. Теперь следует ввести подклассы класса *Пицца\_Основа*. В большом окне текстового редактора введите имена подклассов *ТонкаяИХрустящая* и *ГлубокийПротивень*, как показано на рис. 1.12.

5. Нажмите кнопку «Далее (Continue)» мастера. Мастер будет проверять, правильно ли задано имя, а также уникальность имен – два имени не могут быть одинаковыми. Если есть ошибки в именах классов, они будут описаны на этой же странице, наряду с предложениями по их исправлению.

6. Нажмите кнопку «Далее (Continue)» мастера. Убедитесь, что поле «*Сделать все новые классы непересекающимися*» отмечено галочкой (рис. 1.13), при этом мастер будет автоматически создавать новые классы не пересекающимися друг с другом.



**Рис. 1.12.** Создание иерархии классов: ввод имён подклассов  
ТонкаяИХрустящая и ГлубокийПротивень



**Рис. 1.13.** Объявление классов непересекающимися

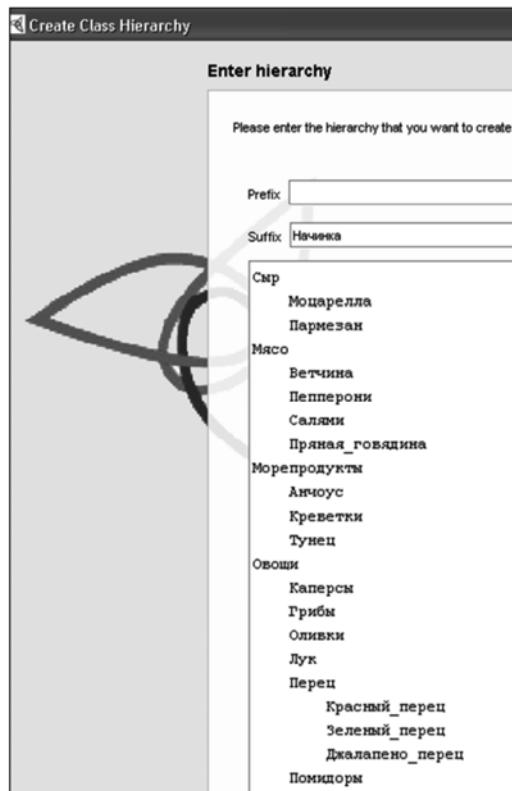
Таким образом, элемент класса *Пицца\_Основа* не может быть одновременно элементом подкласса *ТонкаяИХрустящая* и элементом подкласса *ГлубокийПротивень*.

### **Создание некоторых начинок пиццы**

Теперь у нас есть некоторые базовые классы, давайте создадим начинки пиццы. Начинки будут сгруппированы по различным категориям – мясные начинки, растительные начинки, начинки из сыра и морепродуктов.

**Задание 6:** создайте несколько подклассов класса *Пицца\_Начинка*.

1. Выберите класс *Пицца\_Начинка* в иерархии классов.



**Рис. 1.14.** Иерархия начинок пиццы

2. Воспользуйтесь мастером «*Создать Иерархию классов* (*Create Class Hierarchy*)» таким же образом, как это было сделано в предыдущем задании.

3. Убедитесь, что выбран класс *Пицца\_Начинка*, и нажмите кнопку «*Далее*».

4. Если мы хотим, чтобы имена новых классов имели окончание *Начинка*, необходимо в поле *Suffix* ввести это окончание.

5. В окне текстового редактора задается дерево подклассов. При этом каждый новый класс вводится в новой строке. Если он является подклассом класса, расположенного выше, то это показывается с помощью клавиши табуляция (*Tab*), имя его будет вводиться с отступом вправо. Введите имена классов, как показано на рис. 1.14. Обратите внимание, что с помощью табуляции имена подклассов сдвинуты вправо.

6. Создав с использованием отступа список классов, нажмите кнопку «*Далее*» и убедитесь, что помечен галочкой флажок «*Сделать непересекающимися все примитивные классы одного уровня иерархии*», чтобы новые классы-братья были сделаны не пересекающимися друг с другом.

7. Нажмите кнопку «*Готово*» для создания классов и закройте инструмент.

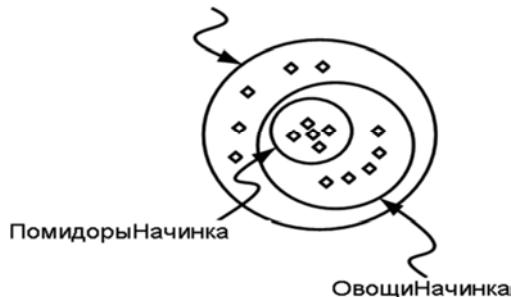
Иерархия классов должна иметь вид, как показано на рис. 1.15. До сих пор мы строили иерархию классов скорее интуитивно. Что же обозначает эта иерархия? Если *ОвоциНачинка* является подклассом *Пицца\_Начинка*, тогда ВСЕ экземпляры *ОвоциНачинка* являются экземплярами *Пицца\_Начинка* без исключения. Если что-то принадлежит классу *ОвоциНачинка*, то это означает, что оно также принадлежит и классу *Пицца\_Начинка*, как показано на рис. 1.16.

Именно по этой причине мы педантично добавили к имени всех начинок окончание «*Начинка*», например *ВетчинаНачинка*. Несмотря на то, что имена классов сами не несут никакой формальной семантики в OWL (и на других языках онтологий), мы внесли эту семантику с помощью имени. *Просто Ветчина* имеет другой смысл – смысл ветчины как мясного продукта.



Рис. 1.15. Иерархия классов

Пицца\_Начинка

Рис. 1.16. Диаграмма, поясняющая разбиение на классы, подклассы, подгруппы класса *Пицца\_Начинка*

Следует напомнить, что иерархия классов (или таксономия) описывается простой логической формулой – импликацией  $A \rightarrow B$ . В этой формуле класс  $A$  является подклассом  $B$ .

### **Свойства и их характеристики**

Свойства описывают отношения. Существуют два основных типа свойств: свойства объектов (*Object Properties*) и свойства определенного типа данных (*Data Properties*). Свойства объектов (*Object Properties*) описывают отношения между двумя индивидуальностями (*объектами*), а свойства данных (*Data Properties*) – связь объекта с характеристикой, значение которой принадлежит определенному типу данных.

В этом разделе основное внимание уделяется *Свойствам объектов* (*Object Properties*), *Свойства данных* (*Data Properties*) описаны в разделе 1.6. OWL также имеет третий вид свойств – *Annotation Properties* (свойства аннотации). *Annotation Properties* могут быть использованы для добавления информации (метаданные – данные о данных) для классов, индивидуальностей и свойств объектов или данных.

Свойства могут быть созданы на вкладке «*Object Properties*», как показано на рис. 1.17. Рис. 1.18 показывает кнопки, расположенные в верхнем левом углу вкладки «*Object Properties*», которые используются для создания свойств объектов (*Object Properties*) и свойств данных определенного типа (*Data Properties*), а также могут быть использованы для *Annotation Properties*.

**Задание 7:** создайте свойство (отношение) *имеетИнгредиенты*.

1. Переключитесь на вкладку «*Object Properties*». Используя кнопку «Добавить свойство» (см. рис. 1.18), создайте новое свойство объектов.

2. Введите имя свойства в диалоговом окне, как показано на рис. 1.19.

Хотя не существует строгого наименования для свойств, рекомендуется имена свойств начинать с маленькой буквы, не допускать пробелов и новые слова в имени начинать с заглавной буквы. Также рекомендуется начинать имя свойства с префикса «имеет» или «является», например *имеетЧасть*, *являетсяЧастью*, *имеетПроизводителя*, *являетсяПроизводителем*. Так как свойства – это бинарные отношения, то можно предложить и второе правило:

имя формировать из двух слов и знака подчеркивания между ними, слова должны называть связываемые этим свойством классы. Например: *студент\_группа* – имя отношения, связывающего элемент класса Студент с элементом класса *Группа*.

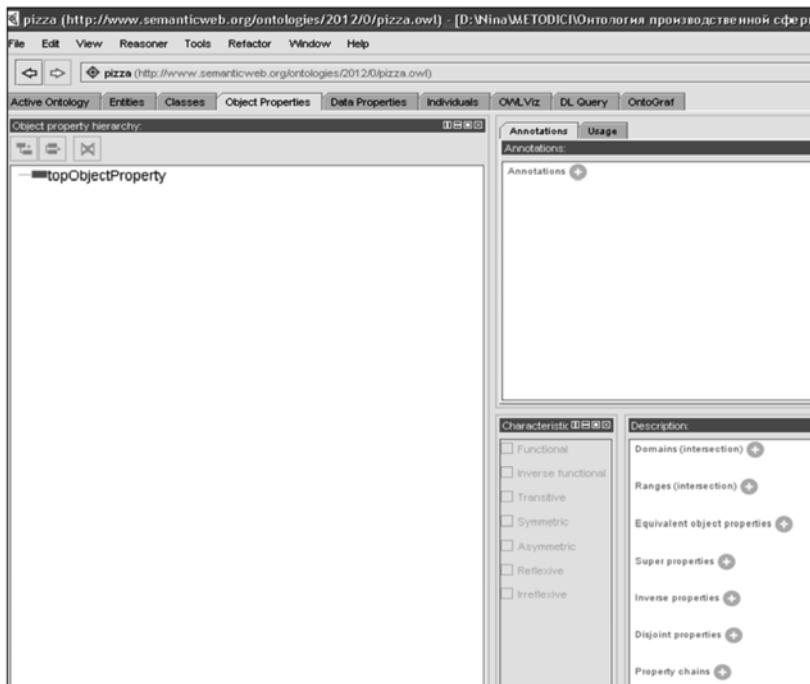


Рис. 1.17. Вкладка *Object Properties*

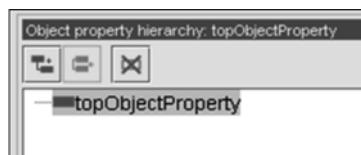
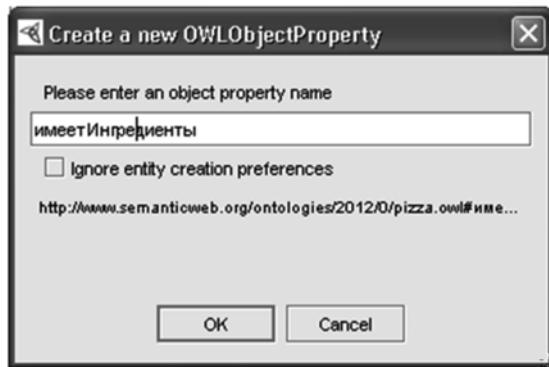


Рис. 1.18. Кнопки для добавления и удаления свойств объектов

После того как было добавлено свойство *имеетИнгредиенты*, можно добавить еще два свойства – *имеетНачинку* и *имеетОснову*. Свойства могут иметь вложенные свойства, так что можно формировать иерархию свойств так же, как и иерархию классов. Нап-

ример, свойство *имеетМаму* может быть наследником свойства *имеетРодителя*.

В случае онтологии пиццы свойства *имеетНачинку* и *имеетОснову* должны быть созданы в качестве подсвойств свойства *имеетИнгредиенты*. Если свойство *имеетНачинку* (или свойство *имеетОснову*) соединяет две индивидуальности, это означает, что эти два объекта также связаны свойством *имеетИнгредиенты*.



**Рис. 1.19.** Диалоговое окно для ввода имени свойства

**Задание 8:** создайте свойства *имеетНачинку* и *имеетОснову* как наследников свойства *имеетИнгредиенты*.

1. Чтобы создать свойство *имеетНачинку* как наследника свойства *имеетИнгредиенты*, выберите на вкладке «Object Properties» в иерархии свойств элемент *имеетИнгредиенты*.

2. Нажмите кнопку «Добавить подсвойство». Новое свойство объекта будет создано как подсвойство элемента *имеетИнгредиенты*.

3. Задайте имя нового свойства *имеетНачинку*.

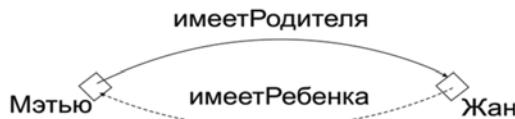
4. Повторите все шаги для создания свойства *имеетОснову*.

Обратите внимание, что возможно создание подсвойств для данных определенного типа (*Data Properties*). Однако не представляется возможным смешивать и сочетать *Object Properties* и *Data Properties* (свойства объекта и свойства данных).

### Обратные (инверсные) свойства

Каждое свойство объекта может иметь соответствующее обратное свойство. Если некоторая сущность *a* связана с сущностью *b*, то сущность *b* так же имеет обратную связь с сущностью *a*. Например, на

рис. 1.20 показаны свойства *имеетРодителя* и его обратное свойство *имеетРебенка* – если Мэтью *имеетРодителя* Жан, то из обратного свойства мы можем сделать вывод, что Жан *имеетРебенка* Мэтью. Другой пример: обратное свойство к отношению *студент\_группа* будет *группа\_студент*.



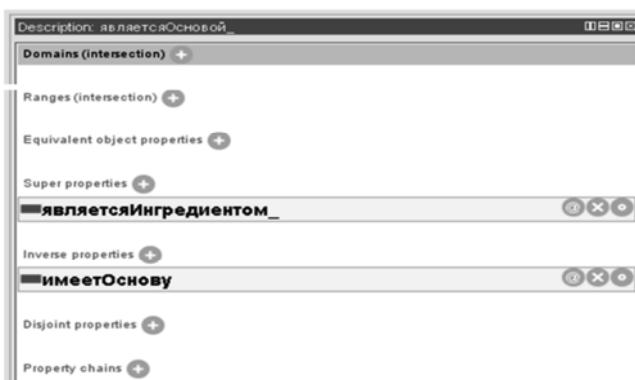
**Рис. 1.20.** Пример обратного свойства: свойство *имеетРебенка* является обратным к свойству *имеетРодителя*

Обратные свойства могут быть созданы / определены в секции *Inverse properties*, как показано на рис. 1.21.

**Задание 9:** создайте некоторые обратные свойства.

1. Используйте кнопку «Добавить свойство» на вкладке «Object Properties», чтобы создать новое свойство с именем *являетсяИнгредиентом* (оно будет обратным свойству *имеетИнгредиенты*).

2. Нажмите «Добавить» значок (+) рядом с заголовком *Inverse properties*, как показано на рис. 1.21. На экране появится диалоговое окно, в котором в дереве свойств могут быть выбраны нужные свойства. Выберите элемент *имеетОснову* и нажмите «OK». Свойство *имеетОснову* должно теперь отображаться как *Inverse properties*.

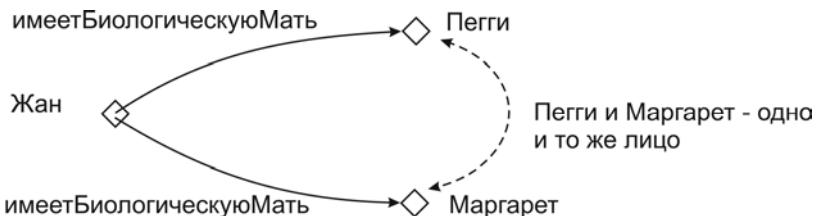


**Рис. 1.21.** Свойство *имеетОснову* является обратным к свойству *явлетсяОсновой\_*

3. Аналогично создайте обратные свойства: являетсяОсновой\_, являетсяНачинкой\_ как подсвойства свойства являетсяИнгредиентом.

### Характеристики свойств объекта

Язык OWL позволяет более точно передать смысл свойств за счет использования характеристик свойства. Далее рассматриваются различные характеристики, которые могут иметь свойства.



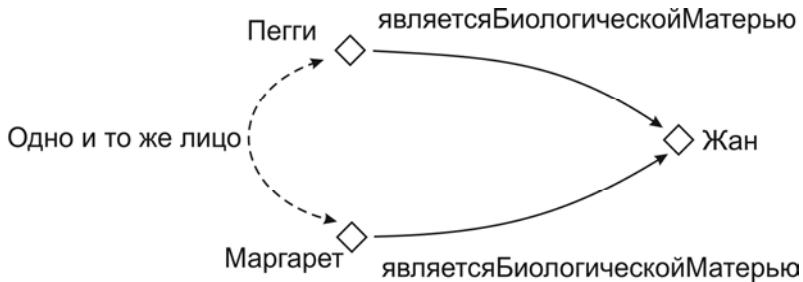
**Рис. 1.22.** Пример функционального свойства:  
*имеетБиологическуюМаму*

### Функциональные свойства

Если свойство функционально, то для данного индивида не может быть более одного объекта, с которым он связан через данное свойство. Рис. 1.22 показывает пример функционального свойства *имеетБиологическуюМаму*. У каждого индивида биологическая мама одна. Если мы говорим, что *Жан имеетБиологическуюМаму Пегги* и мы также говорим, что *Жан имеетБиологическуюМаму Маргарет*, то, поскольку *имеетБиологическуюМаму* – функциональное свойство, мы можем заключить, что *Пегги и Маргарет* должны быть одним и тем же человеком. Следует отметить, однако, что если *Пегги и Маргарет* были явно заданы как различные лица, то вышеуказанное заявление приведет к противоречию.

### Обратные Функциональные свойства

Если свойство обратно-функциональное, то это означает, что обратное свойство также представляет собой функцию. Для данного индивида не может быть более одного объекта, связанного с этим индивидом через свойство. Можно сказать, что обратно-функциональное свойство соответствует степени связи 1:1. Рис. 1.23 показывает пример обратного функционального свойства *являетсяБиологическойМатерью*.

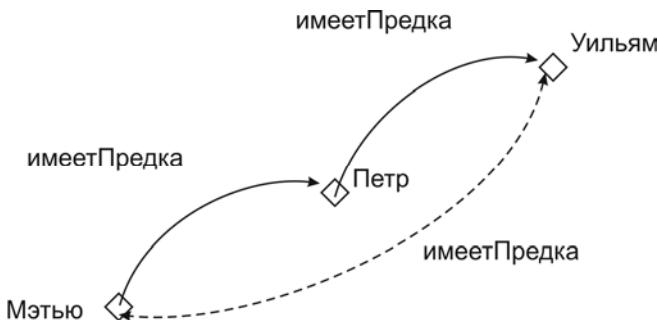


**Рис. 1.23.** Обратное функциональное свойство  
*являетсяБиологическойМатерью\_*

Это обратное свойство к свойству *имеетБиологическуюМать*. Свойство *являетсяБиологическойМатерью* – обратное функциональное. Его смысл: Peggi или Margaret – мама только одного ребенка.

### Транзитивные Свойства

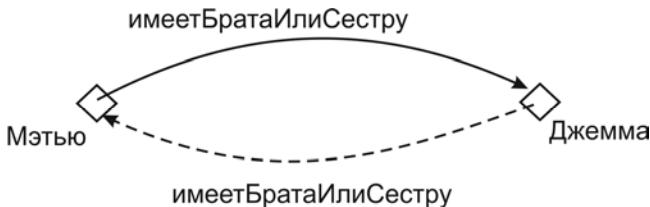
Если свойство транзитивно и связывает отдельные индивиды *a* и *b*, а также индивид *b* с индивидом *c*, то мы можем сделать вывод, что индивид *a* связан этим же свойством с индивидом *c*. Например, рис. 1.24 показывает пример транзитивного свойства *имеетПредка*. Если Мэтью является предком Петра и Петр – предок Уильяма, то можно сделать вывод, что Мэтью – предок Уильяма, – на это указывает пунктирная линия на рис. 1.24.



**Рис. 1.24.** Пример транзитивного свойства: *имеетПредка*

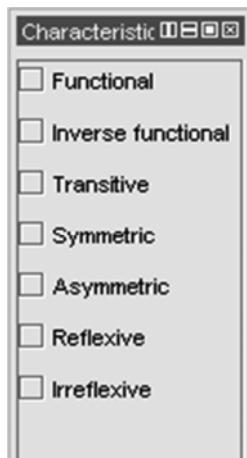
### Симметричные свойства

Если свойство Р является симметричным и свойство связывает индивида *a* с индивидом *b*, то *b* также связан с индивидом *a* с помощью того же свойства Р. Рис. 1.25 показывает пример симметричного свойства. Если индивид *Мэтью* связан с индивидом *Джеммой* через свойство *имеетБратаИлиСестру*, то мы можем заключить, что *Джемма* также должна быть связана с *Мэтью* через свойство *имеетБратаИлиСестру*.



**Рис. 1.25.** Пример симметричного свойства: *имеетБратаИлиСестру*

Можно сделать свойство *имеетИнгредиенты* транзитивным для того, чтобы сказать, если начинка пиццы имеет какие-то ингредиенты, то сама пицца имеет те же ингредиенты. Характеристики свойства задаются в секции *Characteristic* (см. рис. 1.17 и рис. 1.26).



**Рис. 1.26.** Характеристики свойств

**Задание 10:** определите свойство *имеетИнгредиенты* транзитивным.

1. Выберите элемент *имеетИнгредиенты* в дереве свойств.
2. Для флагка «Транзитивный» установите галочку на панели *Characteristik*.
3. Выберите свойство *являетсяИнгредиентом*, которое является обратным *имеетИнгредиенты*. Убедитесь, что поле «Транзитивный» отмечено галочкой. Если свойство транзитивно, то обратное свойство также должно быть транзитивным.

Заметим, что если свойство транзитивно, то оно не может быть функциональным. Причиной этого является то, что транзитивные свойства по своей природе могут образовывать «цепочки» индивидуальностей. Определить транзитивное свойство функциональным, следовательно, не имеет смысла.

Теперь следует сказать, что пицца может иметь только одну основу. Есть множество способов выполнить это. Однако для этого мы сделаем свойство *имеетОснову* функциональным, так что основа будет иметь только одно значение для данной пиццы.

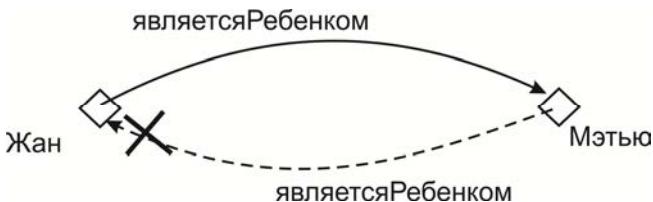
**Задание 11:** определите свойство *имеетОснову* как функциональное

1. Выберите свойство *имеетОснову* .
2. Поставьте галочку в квадрате *Functional* (*Функциональный*)

Если выбрана вкладка *Data Properties*, то панель характеристик сокращена до характеристики *Functional* (*Функциональный*). Это происходит потому, что OWL-DL не позволяет свойствам данных быть транзитивными, симметричными, рефлексивными и обратными.

### **Асимметричные свойства**

Если свойство Р является асимметричным и связывает индивида *a* с индивидом *b*, то индивид *b* не может быть связан с индивидом *a* через свойство Р. Рис. 1.27 показывает пример асимметричного свойства. Если индивид *Жан* связан с индивидом *Мэтью* через свойство *являетсяРебенком*, то можно утверждать, что *Мэтью* не может являться ребенком *Жана*, а потому не может быть связан с *Жаном* свойством *являетсяРебенком*. Это, однако, не мешает *Мэтью* быть связанным с другим лицом *Биллом* через свойство *являетсяРебенком*.



**Рис. 1.27.** Пример асимметричного свойства *являетсяРебенком*

### Рефлексивные свойства

Свойство  $P$  называется рефлексивным, когда это свойство связывает индивида с самим собой. На рис. 1.28 мы видим пример рефлексивного свойства *знает*, Джордж знает Джорджа, т.е. самого себя. Однако, кроме того, Джордж может знать других людей, поэтому Джордж может быть связан отношением *знает* с индивидом Симоном.



**Рис. 1.28.** Пример рефлексивного свойства *знает*

### являетсяМамой



**Рис. 1.29.** Пример иррефлексивного свойства *являетсяМамой*

### Иррефлексивные свойства

Если свойство  $P$  иррефлексивное, то оно может быть описано как свойство, которое связывает индивида  $a$  с индивидом  $b$ , только когда эти индивиды различны. Как пример рассмотрим свойство *являетсяМамой*: индивид Алиса может быть связан с индивидом Бобом свойством *являетсяМамой*, но Алиса не может быть мамой самой себе (рис. 1.29).

### Цепочки свойств (отношений)

Свойства (отношения) можно соединять в цепочки (создавать композицию свойств) и благодаря этому получать новые отно-

шения. Рассмотрим простой пример. Пусть имеются следующие отношения: *являетсяПапой*, *являетсяМамой* и *являетсяРодителем*, причем первые два свойства – подсвойства для *являетсяРодителем*. И пусть некто X *являетсяПапой* Y, а Y *являетсяРодителем* Z, следовательно, X *являетсяДедушкой* Z. Аналогично, если некто X *являетсяМамой* Y, а Y *являетсяРодителем* Z, то X *являетсяБабушкой* и *являетсяБабушкой* на основе базовых отношений. В логике предикатов эти предложения запишутся следующим образом: *являетсяБабушкой(X, Z) ← являетсяМамой(X, Y), являетсяРодителем(Y, Z)*.

*являетсяДедушкой(X, Z) ← являетсяПапой(X, Y), являетсяРодителем(Y, Z).*

**Задание 12:** создайте новое свойство *являетсяБабушкой* на основе двух базовых отношений *являетсяМамой* и *являетсяРодителем*.

1. Переключитесь на вкладку «*Object Properties*». Используя кнопку «*Добавить свойство*» (см. рис. 1.18–1.19), создайте новое свойство *являетсяРодителем* и его подсвойства *являетсяПапой*, *являетсяМамой* так, как показано на рис. 1.30.

2. Убедитесь, что выделен пункт *topObjectProperty* в иерархии свойств. Нажмите кнопку «*Добавить подсвойство (+)*» и введите имя *являетсяБабушкой*.

3. Выберите новое свойство *являетсяБабушкой* в иерархии свойств и опишите его. Для этого на панели «*Описание свойства ...*» в секции *Property chains* (*Цепочка свойств*) нажмите кнопку (+). Откроется диалоговое окно текстового редактора (рис. 1.31), в котором наберите цепочку «*являетсяМамой o являетсяРодителем*». В этом выражении знак «*o*» (английская маленькая буква *o*) обозначает операцию композиции двух отношений. В этом редакторе также можно использовать технику автозаполнения.

В результате секция *Property chains* (*Цепочка свойств*) будет иметь вид, показанный на рис. 1.30.

Аналогично можно создать отношение *являетсяДедушкой*. Чтобы создать свойство *являетсяПрадедушкой* нужно использовать цепочку (композицию) из трех отношений: «*являетсяПапой o являетсяРодителем o являетсяРодителем*».

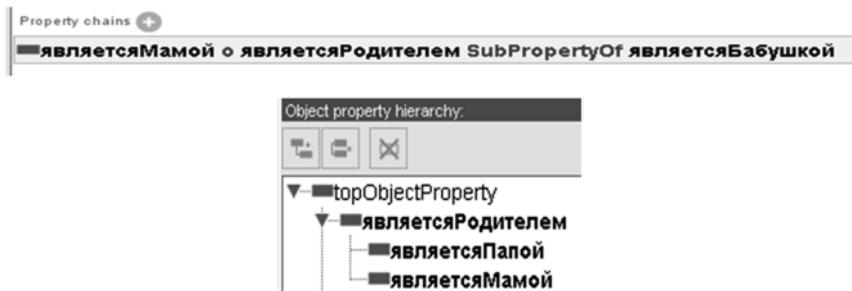


Рис. 1.30. Базовые отношения

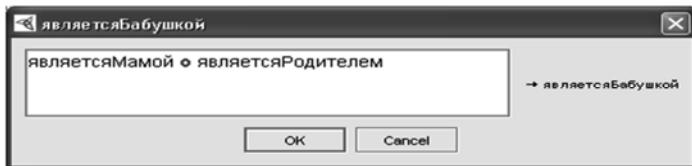


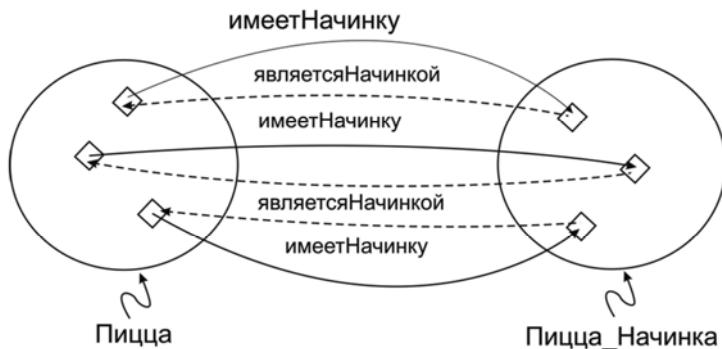
Рис. 1.31. Описание нового свойства как цепочки базовых свойств

С помощью вновь полученных свойств можно создавать такие классы, как «Бабушки», «Дедушки» и «Прадедушки».

Приведем еще примеры. На основе базовых отношений *студент\_группа*, *группа\_факультет* можно получить производное отношение *студент\_факультет*, описав его как цепочку «*студент\_группа o группа\_факультет*». Для онтологии Пиццы, создав цепочку «*имеетНачинку o имеетПряность*», можно получить новое отношение «*пицца\_пряность*».

### Домены и диапазоны свойств

Свойство описывает бинарное отношение. Доменом называется область определения отношения, а диапазоном – область значений отношения. Свойства (отношения) связывают индивидов из домена с индивидами диапазона. Например, в онтологии пиццы свойство *имеетНачинку* связывает объекты, принадлежащие к классу *Пицца*, с объектами, принадлежащими к классу *Пицца\_Начинка*. В этом случае – домен свойства *имеетНачинку* класс *Пицца*, а диапазон – *Пицца\_Начинка*. Это изображено на рис. 1.32.



**Рис. 1.32.** Домен и диапазон для свойства *имеетНачинку* и обратного ему свойства *являетсяНачинкой*

**Домены и диапазоны свойств в OWL.** Важно понимать, что в OWL домены (*Domains*) и диапазоны (*Ranges*) используются для проверки корректности онтологии. Они используются в качестве «аксиом» при выводе. Например, если у свойства *имеетНачинку* установлен доменом класс *Пицца*, а мы затем применяем *имеетНачинку* к элементам класса *Мороженое*, то это, как правило, приведет к ошибке. Ошибка будет возникать, если *Пицца* не пересекается с *Мороженым*. И ошибки не будет в случае, если *Мороженое* подкласс *Пиццы*!

Пусть требуется указать, что отношение *имеетНачинку* связывает элементы класса *Пицца* с целым рядом индивидуальностей класса *Пицца\_Начинка*. Для этого можно определить диапазон, как показано на рис. 1.33.



**Рис. 1.33.** Домен и диапазон свойства *имеетНачинку*

**Задание 13:** Создайте диапазон (*Range*) для свойства *имеетНачинку*

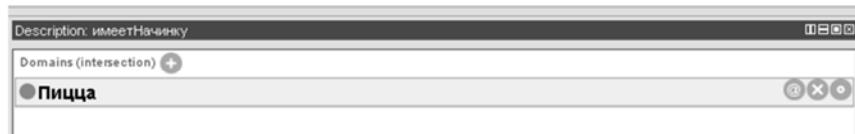
1. Убедитесь, что свойство *имеетНачинку* выбрано в иерархии свойств на вкладке «*Object Properties*».

2. Нажмите «Добавить» значок (+) рядом с «*Ranges*» (рис. 1.33). Появится диалоговое окно, в котором выберите класс *Пицца\_Начинка*, являющийся диапазоном для свойства *имеетНачинку*.

3. Выберите *Пицца\_Начинка* и нажмите кнопку «OK». *Пицца\_Начинка* теперь появится в списке диапазонов отношения.

Можно указать несколько классов в качестве диапазона для свойства. Если указано несколько классов, то диапазон интерпретируется как пересечение классов. Например, если диапазон имеет классы *Мужчина* и *Женщина*, то результатом будет пересечение этих классов, т.е. пустое множество.

На рис. 1.34 и в задании 14 показано, как определить домен свойства.



**Рис. 1.34.** Домен отношения *имеетНачинку*

**Задание 14:** определите домен (*Domains*) для свойства *имеетНачинку*.

1. Убедитесь, что свойство *имеетНачинку* выбрано в иерархии свойств на вкладке «*Object Properties*».

2. Нажмите «Добавить» – значок (+) рядом с «*Домен (Domains)*» (рис. 1.34). Появится диалоговое окно, в котором выберите класс *Пицца*, являющийся доменом для свойства *имеетНачинку* (рис. 1.34).

Это означает, что индивиды, которые используются как аргументы на левой стороне отношения *имеетНачинку*, будут являться членами класса *Пицца*. А любые объекты, которые используются как аргументы на правой стороне отношения *имеетНачинку*, будут являться членами класса *Пицца\_Начинка*. Аналогично укажите домен и диапазон для свойств *имеетОснову* и *являетсяОсновой\_*.

Выше было описано, как указывать домены и диапазоны для отношений. Это было сделано в целях познакомить с различными возможностями редактора *Protege* и языка OWL. Однако это делать

не рекомендуется, особенно для больших онтологий. Указание условий в виде домена и диапазона может привести к «неожиданным» результатам классификации.

Выполнив задания этого раздела пособия, вы научились создавать классы, подклассы, объявлять их непересекающимися, а также определять свойства или бинарные связи между объектами, познакомились с характеристиками свойств и их смыслом. Следующие разделы посвящены способам описания классов с помощью аксиом.

### **Контрольные вопросы**

1. Что такое онтология?
2. Назовите основные компоненты онтологии и дайте им определение.
3. Для чего нужно уметь разрабатывать онтологию? Где онтология может быть использована?
4. Как создать дерево классов в Protege 4? Как при этом использовать мастера?
5. Какие виды отношений (свойств) в Protege 4 вы знаете?
6. Как создать свойство (отношение)?
7. Что такое обратное свойство?
8. Что такое транзитивное отношение? Что такое рефлексивное отношение?
9. Как с помощью цепочки свойств создать новое свойство?
10. Что такое домен и диапазон свойства?

### **1.3. Определение и описание классов**

Созданные отношения (свойства) можно теперь использовать для определения и описания классов онтологии пиццы.

#### **Описание классов с помощью ограничений на связи объектов**

Напомним, что в OWL свойства описывают бинарные отношения. Свойства данных (*Data Properties*) описывают отношения между объектами и значениями их атрибутов, задаваемыми данными определенного типа. Свойства объектов (*Object Properties*) описывают отношения между двумя индивидами. Например, на рис. 1.2 индивид *Мэтью* связан с индивидом *Джеммой* через свойство *имеетБратаИлиСестру*. Теперь рассмотрим всех индиви-

дов, которые участвуют в связи *имеетБратаИлиСестру*. Можно считать, что эти люди принадлежат к категории лиц, у которых есть брат или сестра. Ключевая идея состоит в том, что эту категорию лиц можно описать как класс, используя ограничения, построенные на основе отношений. Ограничение описывает класс объектов на основе отношений и может называться утверждением.

### **Примеры ограничений**

Давайте рассмотрим несколько примеров описания классов на основе их связей (свойств). Рассмотрим следующие категории:

- категория лиц, которые имеют хотя бы одного брата или сестру;
- категория лиц, которые имеют брата или сестру и среди них есть хотя бы один брат (мужчина);
- класс людей, у которых есть только сестры (женщины);
- категория лиц, которые имеют более трех братьев или сестер;
- категория пицц, которые имеют в качестве начинки хотя бы один вид начинки, относящейся к классу *МоцареллаНачинка*;
- класс пицц, у которых есть *только* начинки с растительными наполнителями, т.е. начинки – элементы класса *ОвоциНачинка*.

В OWL можно описать все вышеперечисленные классы индивидов с использованием ограничений. Ограничения в OWL можно разделить на три основные категории.

1. Ограничения – Квантификаторы (Кванторы).
2. Ограничения мощности.
3. *HasValue* ограничения.

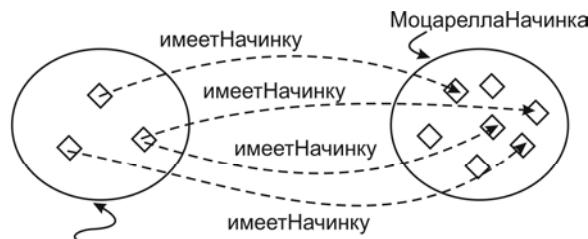
Сначала будем использовать ограничения – кванторы, которые могут быть далее разделены на экзистенциальные ограничения и универсальные ограничения. Оба типа ограничений будут проиллюстрированы примерами.

### **Экзистенциальные и универсальные ограничения**

Экзистенциальные ограничения описывают классы индивидов, которые участвуют, по крайней мере, в одной заданной связи с индивидами указанного класса. Например, «Категория пицц, которые имеют в качестве начинки хотя бы один вид начинки, относящейся к классу *МоцареллаНачинка*». В *Prot'eg'e 4* ключевое слово *some* (некоторые) используется для обозначения экзистенциаль-

ных ограничений. Экзистенциальные ограничения соответствуют квантору существования ( $\exists$ ) в логике. Они также известны как «*someValuesFrom*» ограничения в OWL.

Универсальные ограничения описывают классы индивидов, которые участвуют в заданной связи **только (only)** с индивидами указанного класса. Например, «класс пицц, которые имеют в качестве начинки только начинки, являющиеся членами класса *ОвощиНачинка*». Универсальные ограничения соответствуют квантору всеобщности ( $\forall$ ) в логике. Они также известны как «*allValuesFrom*» ограничения в OWL.



Сущности, которые имеют, по крайней мере, одну начинку типа *МоцареллаНачинка*  
«имеетНачинку some *МоцареллаНачинка*»

Рис. 1.35. Ограничение «имеетНачинку *some* *МоцареллаНачинка*»

Рассмотрим пример экзистенциального ограничения: «имеетНачинку *some* *МоцареллаНачинка*». Это ограничение можно проиллюстрировать с помощью рис. 1.35 – квадратики на рисунке представляют объекты. Ограничение описывает анонимный класс (безымянный класс). Анонимный класс содержит все пиццы, которые удовлетворяют ограничению, то есть все пиццы, которые в качестве начинки имеют хотя бы один вид начинки, относящейся к классу *МоцареллаНачинка*.

Ограничения класса можно просматривать и редактировать, используя панель «*Описание (Description) класса*» вкладки *Классы*, которая приведена на рис. 1.36. Панель «*Описание класса*» является сердцем вкладки *Классы* в Prot'eg'e, и имеет практически всю информацию для описания класса. На первый взгляд, панель «*Описание класса*» может показаться сложной, однако это невероятно мощный способ описания и определения классов.

## Экзистенциальные ограничения

Экзистенциальные ограничения являются на сегодняшний день самым распространенным типом ограничений в онтологии OWL. Экзистенциальное ограничение описывает категории объектов, которые имеют заданную связь хотя бы с одним (или несколькими) индивидами указанного класса. Например, «имеетОснову **some** Пицца\_Основа» описывает все пиццы, которые имеют хотя бы одну основу, являющуюся членом класса Пицца\_Основа.

**Задание 1:** добавьте ограничение к классу *Пицца*, которое определяет, что *Пицца* должна иметь основу – объект класса *Пицца\_Основа*.

1. Выберите *Пицца* из иерархии классов на вкладке «Классы».

2. Выберите «Добавить» – значок (+) рядом с заголовком «Суперкласс (Superclasses)» в панели «Описание класса».

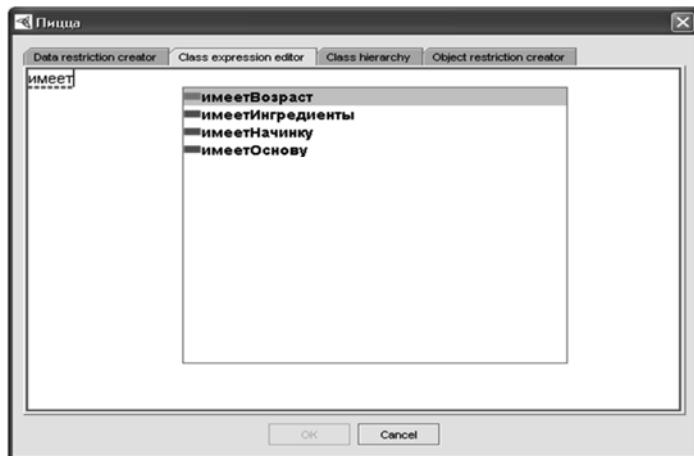
3. Нажмите кнопку «Добавить (+)» как показано на рис. 1.36.

Это позволит открыть диалоговое окно редактора ограничений, где можно ввести ограничение, как показано на рис. 1.37 и рис. 1.38. Текстовое окно позволяет создать ограничение с помощью имен классов, свойств и индивидуальностей. Вы можете перетаскивать классы, свойства и индивидуальности (рис. 1.38) в текстовое поле или вводить (рис. 1.37) их в текстовое поле. При вводе осуществляется проверка всех введенных значений и оповещение о любых ошибках. Для ввода ограничений нужно сделать следующее:

- ввести нужное свойство или выбрать его из списка свойств.
- ввести тип ограничения из списка типов, например «**some**», для экзистенциального ограничения.
- указать требуемые классы как аргументы ограничения.



Рис. 1.36. Создание необходимых ограничений



**Рис. 1.37.** Создание экзистенциального ограничения  
(Пример автозаполнения).

Если будут ошибки, которые подчеркнуты ярким цветом в текстовом поле, то еще раз проверьте, что тип ограничения, свойство и класс были указаны правильно. Очень полезная возможность – построитель выражений. Он предоставляет функцию «автозаполнения» имен классов, имен свойств и индивидуальностей. Автозаполнение активируется при нажатии на кнопку «*Alt Tab*» или «*Ctrl-Пробел*» на клавиатуре. В приведенном выше примере, если мы ввели *Пи* в редактор выражений и нажали клавишу табуляции, выбор для завершения слова *Пи* будет извлекаться из списка, как показано на рис. 1.38. Вверх и вниз клавиши со стрелками могут затем быть использованы для выбора *Пицца\_Основа*, и нажатие клавиши ввода завершит слово. Панель «Описание класса» должна теперь выглядеть так, как показано на рис. 1.39.

Для того чтобы что-то было пиццей, необходимо чтобы это что-то имело хотя бы одну основу. *Пицца* – подкласс *Thing*, элементы которого имеют, по крайней мере, одну основу, принадлежащую классу *Пицца\_Основа*. Это показано на рис. 1.40. Когда ограничения используются для описания классов, то эти классы описываются как *анонимные суперклассы* выделенного класса.

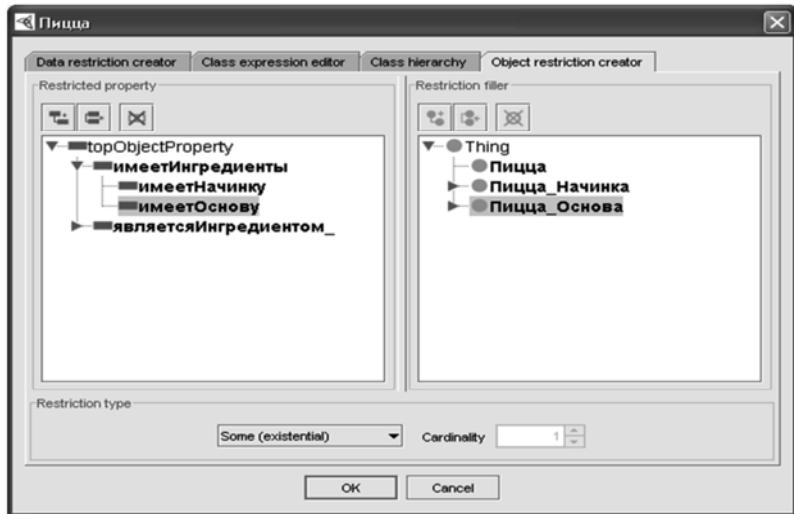


Рис. 1.38. Процесс создания экзистенциального ограничения

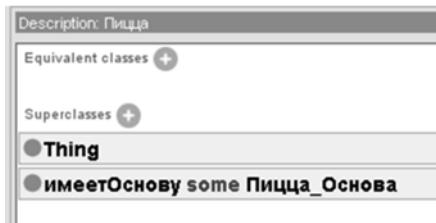


Рис. 1.39. Панель «Описание класса» после ввода ограничения

Пицца

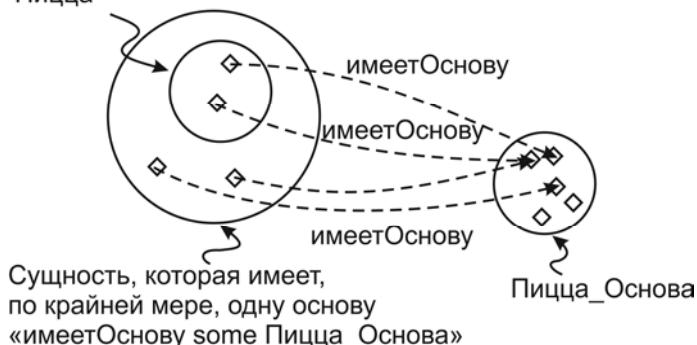


Рис. 1.40. Схема описания Пиццы

## Создание некоторых различных видов пиццы

Пришло время добавить некоторые различные виды пиццы к нашей онтологии. Начнем с добавления вида «*МаргаритаПицца*», который представляет пиццу с начинкой из моцареллы и помидоров. Для того чтобы не нарушать онтологию, создадим группу различных пицц в разделе (классе) «*ИменованнаяПицца*». Для этого создадим подкласс класса *Пицца* с именем *ИменованнаяПицца* и подкласс класса *ИменованнаяПицца* с названием *МаргаритаПицца*.

Добавим комментарий к классу *МаргаритаПицца* с использованием панели «*Аннотация*», расположенной рядом с иерархией классов: «*Пицца, которая имеет начинку только Моцарелла или Томатную*». Это хорошая идея – всегда документировать классы, свойства и т.п. во время редактирования онтологии для того, чтобы просто было общаться с другими разработчиками онтологий.

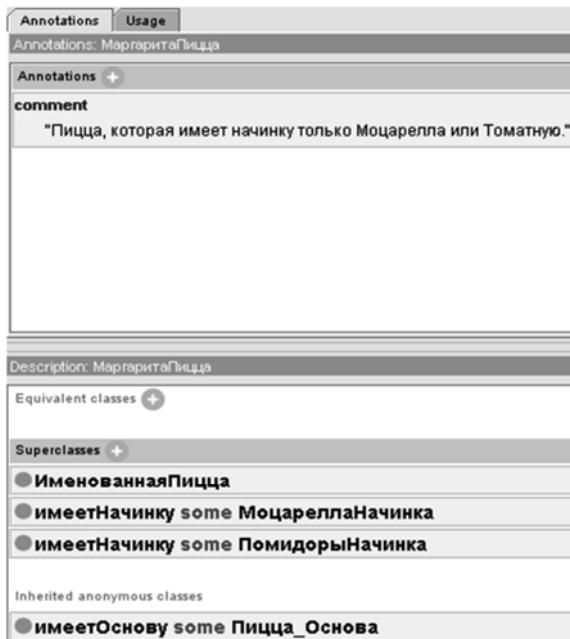


Рис. 1.41. Описание необходимых условий для класса *МаргаритаПицца*

Создав класс *МаргаритаПицца*, мы должны указать начинку, которую она имеет. Для этого мы добавим два ограничения:

*МаргаритаПицца* имеет начинки типа *МоцареллаНачинка* и типа *ПомидорыНачинка*.

**Задание 2:** создайте экзистенциальные (*some*) ограничения для класса *МаргаритаПицца* с использованием свойства *имеетНачинку* и указанием наполнителя из *МоцареллаНачинка*. При этом надо указать, что *МаргаритаПицца* имеет, по крайней мере, одну начинку типа *МоцареллаНачинка*.

1. Убедитесь, что элемент *МаргаритаПицца* выбран в иерархии классов.

2. Выберите «Добавить» – значок (+) рядом с заголовком «*Суперкласс (Superclasses)*» в панели «Описание класса».

3. Наберите *имеетНачинку* как свойство для ограничения в текстовом поле.

4. Наберите «*some*» для создания экзистенциального ограничения.

5. Наберите имя класса *МоцареллаНачинка*. Этого можно добиться, либо введя имя класса *МоцареллаНачинка* в окне редактора, либо с помощью выбора в иерархии классов.

6. Нажмите «*Enter*», чтобы создать ограничение; если есть любые ошибки, ограничение не будет создано и ошибка будет выделена ярким цветом.

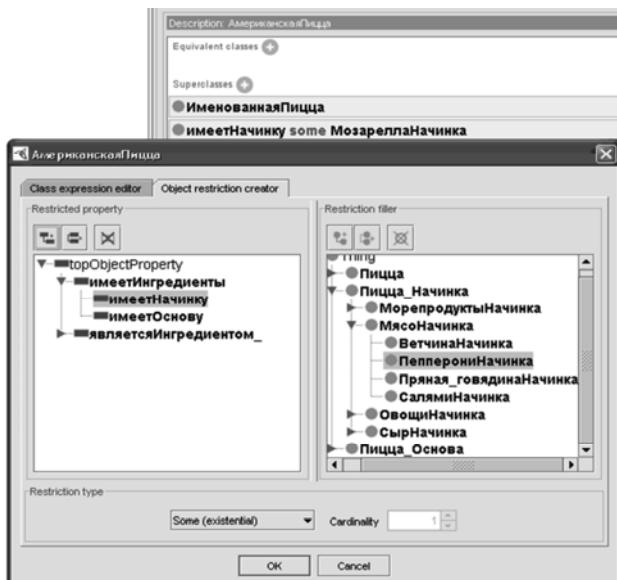
Теперь аналогично укажите, что *МаргаритаПицца* *имеетНачинку ПомидорыНачинка* с помощью ограничения «*имеетНачинку some ПомидорыНачинка*».

Мы добавили ограничения на класс *МаргаритаПицца*, чтобы сказать, что *МаргаритаПицца* – это *ИменованнаяПицца*, что она имеет хотя бы одну начинку вида *МоцареллаНачинка* и хотя бы одну – вида *ПомидорыНачинка*. Более формально, если какой-то элемент принадлежит классу *МаргаритаПицца*, то он также является членом класса *ИменованнаяПицца* и членом *анонимного* класса объектов, элементы которого имеют, по крайней мере, одну связь типа *имеетНачинку* с членом класса *МоцареллаНачинка* и, по крайней мере, одну связь *имеетНачинку* с членом класса *ПомидорыНачинка*.

Теперь создадим класс для описания пиццы с именем *АмериканскаяПицца*, которая имеет начинки – пепперони, сыр моцарелла.

**Задание 3:** создайте вид *АмериканскаяПицца* путем клонирования и изменения описания класса *МаргаритаПицца*.

1. Выберите класс *МаргаритаПицца* в иерархии классов на вкладке *Классы*.
2. Выберите «*Дубликат выделенного класса (Duplicate selected class)*» в меню «*Edit*». Появится диалоговое окно, в котором надо задать имя *АмериканскаяПицца* нового класса и согласиться с тем, что все ограничения будут унаследованы от *МаргаритаПицца*.
3. Убедитесь, что *АмериканскаяПицца* все еще выбрана, нажмите кнопку «*Добавить*», значок (+) рядом с заголовком «*Суперкласс*», чтобы добавить новые ограничения, описывающие необходимые условия для класса *АмериканскаяПицца* (рис. 1.42).
4. Наберите свойство *имеетНачинку*.
5. Наберите *some* для создания экзистенциального ограничения.
6. В качестве аргумента отношения задайте класс *ПепперониНачинка* либо введя его имя в текстовом поле, либо с помощью перетаскивания из дерева классов (рис. 1.42).
7. Нажмите *OK* для создания ограничения.



**Рис. 1.42.** Создание вида *АмериканскаяПицца* путем клонирования и изменения описания *МаргаритаПицца*

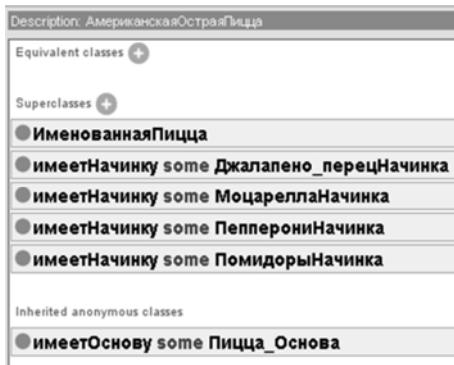


Рис. 1.43. Создание вида *АмериканскаяОстряяПицца*

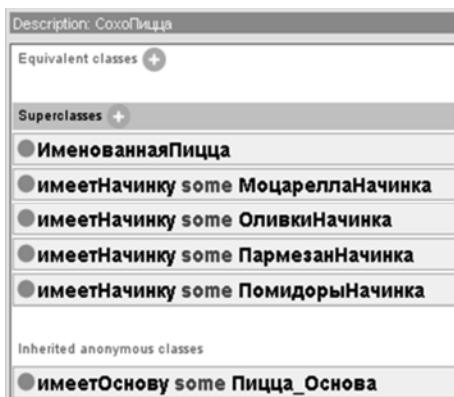


Рис. 1.44. Создание вида *СохоПицца*

**Задание 4:** создайте виды *АмериканскаяОстряяПицца* и *СохоПицца*.

1. *АмериканскаяОстряяПицца* почти такая же, как *АмериканскаяПицца*, но на ней есть *Джалапено* – перцы. Ее можно создать путем клонирования класса *АмериканскаяПицца* и добавить ограничение по свойству *имеетНачинку* с наполнителем из *Джалапено\_перецНачинка*.

2. *СохоПицца* почти такая, как *МаргаритаПицца*, но имеет дополнительные начинки сыр *Пармезан* и оливки. Создать это можно путем клонирования *МаргаритаПицца* и добавления двух эксписиентиальных ограничений по свойству *имеетНачинку*; одно с наполнителем *ОливкиНачинка* и одно с наполнителем из *ПармезанНачинка*.

зан`Начинка`. Для пиццы `АмериканскаяОстнаяПицца` описание класса будет иметь вид, показанный на рис. 1.43. Для `СохоПицца` описание класса изображено на рис. 1.44.

Создав эти пиццы, мы теперь должны сделать их не пересекающимися друг с другом.

**Задание 5:** определите подклассы класса `ИменованнаяПицца` непересекающимися друг с другом.

1. Выберите класс `МаргаритаПицца` в иерархии классов на вкладке «Классы».

2. Выберите в меню «Edit» опцию «Сделать непересекающимися примитивные классы одного уровня (Make primitive siblings disjoint)», чтобы элементы этих классов не пересекались друг с другом.

### **Контрольные вопросы**

1. Чем отличаются Свойства Объектов (Object Property) от свойств данных (Data Property)?

2. Какие виды ограничений вы знаете?

3. Что такое экзистенциальные ограничения?

4. Что такое универсальные ограничения?

5. Как создать экзистенциальные ограничения?

6. Как создать клон для заданного класса? Что при этом произойдет?

7. Как объявить классы непересекающиеся?

8. Какому квантору соответствуют экзистенциальные ограничения?

## **1.4. Использование резонера (машины вывода)**

### **Назначение машины вывода (резонера)**

Одной из ключевых особенностей онтологий, которые описываются с помощью OWL, заключается в том, что они могут быть обработаны резонером (машиной вывода). Одно из предназначений резонера в том, чтобы проверить, действительно ли один класс является подклассом другого класса. При этом выполняется вывод иерархии классов онтологии. Еще одна полезная функция резонера – это с помощью автоматических рассуждений (процесса вывода) провести проверку согласованности ограничений. На основе описания класса можно проверить, возможны или нет заданные

ограничения для класса. Класс считается противоречивым, если его ограничениям не удовлетворяет ни одна сущность. Резонер (машину вывода) иногда называют классификатором. Таким образом, резонер классифицирует объекты предметной области и проверяет согласованность онтологии.

### **Вызов Резонера (Reasoner)**

*Prot'eg'e 4* позволяет подключать различные резонеры к редактору. Резонер, поставляемый с *Prot'eg'e*, называется *Fact++*. Резонер может осуществлять вывод иерархии классов и проверять логическую непротиворечивость онтологии. В *Prot'eg'e 4* та иерархия классов, которая построена вручную, называется *присоединенной иерархией*. Иерархия классов, которая автоматически вычисляется в процессе рассуждений, называется *выводимой иерархией*.

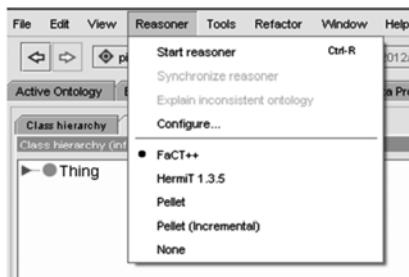


Рис. 1.45. Запуск резонера (машины вывода)

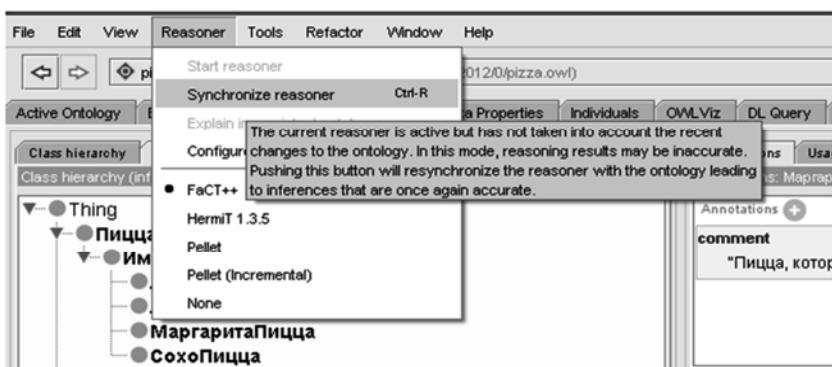


Рис. 1.46. Запуск резонера после введенных  
в онтологию изменений Резонер → Synchronize reasoner

Чтобы автоматически классифицировать онтологию (и проверить ее на непротиворечивость), нужно использовать подпункт «*Start resoner*» или *Synchronize resoner* пункта *Резонер (Reasoner)* главного меню, как показано на рис. 1.45 и рис. 1.46. Если класс был признан несовместимым, то он будет выделен ярким цветом.

### **Несогласованные (противоречивые) классы**

Для того чтобы продемонстрировать использование рассуждений в обнаружении несоответствий в онтологии, создадим новый класс как подкласс *СырНачинка*, а также как подкласс *ОвощиНачинка*. Эта стратегия часто используется для проверки, правильно ли построена онтология. Класс, который добавляется для проверки целостности онтологии, называют тестовым классом.

**Задание 1:** добавьте тестовый класс и назовите его *ТестНепоследовательностиНачинка*, который будет подклассом обоих классов *СырНачинка* и *ОвощиНачинка*.

1. Выберите класс *СырНачинка* из иерархии классов на вкладке *Классы*.

2. Создайте подкласс класса *СырНачинка* с именем *ТестНепоследовательностиНачинка*.

3. Добавьте комментарий к классу *ТестНепоследовательностиНачинка*: «Этот класс должен оказаться несовместимым при проверке онтологии на непротиворечивость». Комментарий позволит любому, кто смотрит на онтологию пиццы, видеть, что мы намеренно ввели противоречивый класс.

4. Убедитесь, что класс *ТестНепоследовательностиНачинка* выбран в иерархии классов, а затем выберите секцию «Суперкласс» в панели «Описание класса».

5. Нажмите на кнопку «Добавить» (+). Появится диалоговое окно (рис. 1.47), содержащее иерархию классов, из которой могут быть выбраны нужные классы. Выберите класс *ОвощиНачинка*, а затем нажмите кнопку «OK». Панель «Описание класса» будет иметь вид, представленный на рис. 1.48.

Если мы рассмотрим иерархию классов, *ТестНепоследовательностиНачинка* должен выглядеть как подкласс *СырНачинка* и как подкласс *ОвощиНачинка*. Это означает, что все индивиды класса *ТестНепоследовательностиНачинка* являются индивидами класса *СырНачинка* и индивидами класса *ОвощиНачинка*. Интуи-

тивно это неверно, так как некоторые сущности не могут быть одновременно и сыром, и овощем!

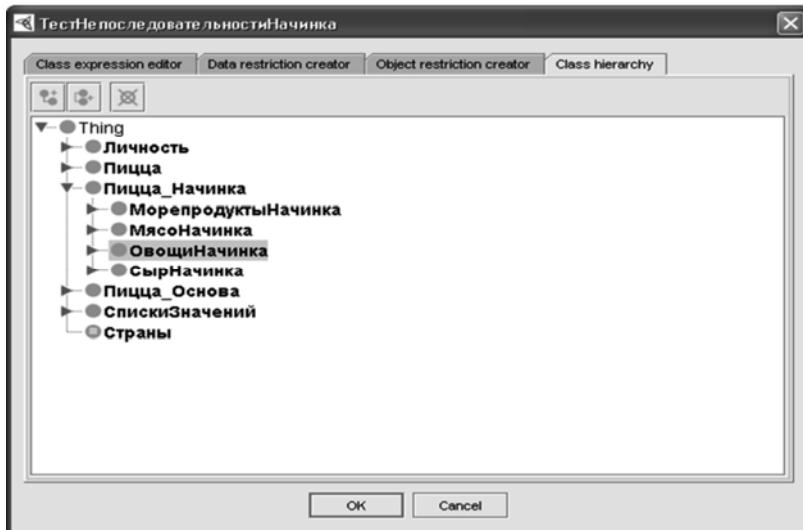


Рис. 1.47. Диалоговое окно редактора выражений

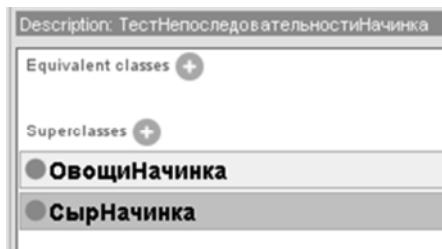


Рис.1.48. Панель «Описание класса *TestНепоследовательностиНачинка*»

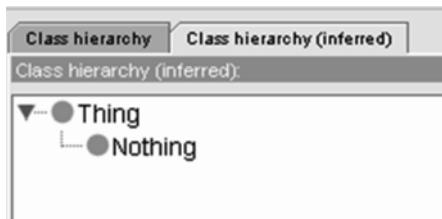


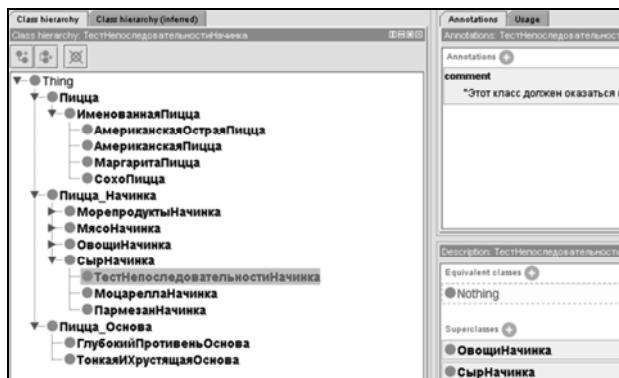
Рис. 1.49. Панель выводимой иерархии до запуска резонера

**Задание 2:** запустите резонер для проверки противоречивости онтологии.

1. Нажмите кнопку «Старт резонер» в выпадающем меню пункта *Резонер* главного меню. Через несколько секунд выводимая иерархия будет вычислена, и ее можно увидеть на вкладке выводимой иерархии. До запуска резонера выводимая иерархия имеет вид, представленный на рис. 1.49, в результате работы машины вывода будет получена иерархия, приведенная на рис. 1.50 и рис. 1.51.

Обратите внимание, что класс *ТестНепоследовательностиНачинка* выделен ярким цветом, указывая на то, что резонер нашел этот класс несовместимым с онтологией (т.е. он не может иметь индивидов в качестве членов класса) (рис. 1.51).

Почему это произошло? Интуитивно мы знаем, что сущность не может в одно и то же время быть сыром и овощем. Сущность не должна быть и экземпляром *СырНачинка*, и экземпляром *ОвощиНачинка*. Однако резонер, опираясь только на имена наших классов, не смог бы найти противоречия в онтологии. Дело в том, что ранее классы *СырНачинка* и *ОвощиНачинка* были объявлены непересекающимися, следовательно, экземпляр класса *СырНачинка* никак не может быть экземпляром класса *ОвощиНачинка* и наоборот. Отсюда *ТестНепоследовательностиНачинка* является пустым классом. И только благодаря тому, что соответствующие классы были объявлены непересекающимися, резонер нашел противоречие в онтологии.



**Рис. 1.50.** Присоединенная иерархия.

Класс *ТестНепоследовательностиНачинка* признан резонером несовместимым с онтологией

Чтобы закрыть выводимую иерархию, используйте маленький белый крест на сером фоне – кнопка в верхней правой части окна выводимой иерархии (рис. 1.51).

**Задание 3:** удалите объявление классов *СырНачинка* и *ОвощиНачинка* непересекающимися классами и посмотрите, что произойдет.

1. Выберите класса *СырНачинка* в иерархии классов.
2. Область *Disjoint classes* содержит классы *ОвощиНачинка*, *МясоНачинка*, *МорепродуктыНачинка*.
3. Нажмите кнопку *Редактировать (Edit)* и удалите из списка класс *ОвощиНачинка*.
4. Выполните классификацию, для этого выберите пункт *Synchronise Reasoner (Синхронизировать резонер)* в выпадающем меню из пункта *Резонер* главного меню.



**Рис. 1.51.** Выводимая иерархия классов после старта резонера

Следует заметить, что *ТестНепоследовательностиНачинка* больше не является несовместимым! Это означает, что экземпляры, которые являются членами класса *ТестНепоследовательностиНачинка*, также являются членами класса *СырНачинка* и *ОвощиНачинка* – сущность может быть и сыром, и овощем!

Это наглядно иллюстрирует важность бережного использования аксиом непересекающихся классов в OWL. В OWL классы, пока они не объявлены как не пересекающиеся друг с другом, по умолчанию считаются пересекающимися. Если неправильно (не в соответствии с предметной областью) описать для классов свойство быть непересекающимися, то это может привести к неожиданным результатам.

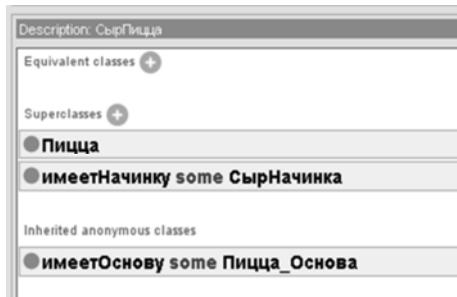
Вернитесь к прежнему варианту онтологии, установив классы *СырНачинка*, *ОвощиНачинка* не пересекающимися друг с другом. Выполните классификацию с помощью резонера и убедитесь в том, что класс *ТестНепоследовательностиНачинка* опять выделен ярким цветом, что говорит о том, что онтология противоречива. Удалите тестовый класс *ТестНепоследовательностиНачинка*.

### Необходимые и достаточные условия

Все классы, которые были созданы до сих пор, удовлетворяют только *необходимым* условиям. *Необходимые* условия можно читать так: «Если сущность является членом этого класса, то она соответствует указанным условиям». При этом мы не можем сказать, что, «если сущность удовлетворяет этим условиям, то она должна быть членом этого класса». Класс, который удовлетворяет только *необходимым* условиям, известен как *примитивный класс*. Проиллюстрируем это на примере.

Мы создадим подкласс пиццы с названием *СырПицца*. Это будет пицца, которая имеет хотя бы один вид начинки типа *СырНачинка*.

**Задание 4:** создайте подкласс пиццы с названием *СырПицца* и укажите, что он имеет, по крайней мере, одну начинку типа *СырНачинка*.



**Рис. 1.52.** Описание *СырПицца* (Использование Необходимых условий) как пиццы, имеющей хотя бы один вид *СырНачинка*

1. Выберите *Пицца* в иерархии классов на вкладке «Классы».
2. Нажмите «Добавить» – значок (+), чтобы создать подкласс пиццы. Назовите его *СырПицца*.
3. Убедитесь, что элемент *СырПицца* выбран в иерархии классов. Нажмите «Добавить» – значок (+) рядом с заголовком «*Суперкласс*».
4. В открывшемся окне редактора наберите выражение «имеетНачинку *some СырНачинка*», которое описывает эзистенциальное ограничение. В завершение нажмите «Enter», чтобы закрыть диалоговое окно и создать ограничение. Панель «Описание класса» должна иметь вид, показанный на рис. 1.52.

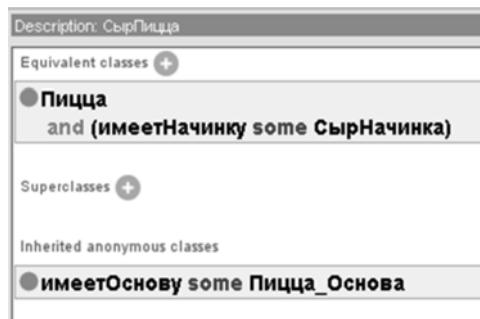
Наше описание *СырПицца* утверждает, что если сущность является членом класса *СырПицца*, то как необходимое условие она принадлежит классу *Пицца* и имеет хотя бы одну начинку типа *СырНачинка*.

Описанию класса в разделе «*Суперкласс*» соответствует логическая формула *СырПицца* → *Пицца and имеетНачинку some СырНачинка*. В этой формуле *СырПицца* подкласс анонимного класса *Пицца and имеетНачинку some СырНачинка*, → – знак операции импликации.

Теперь рассмотрим некоторую (случайно выбранную) сущность. Предположим, известно, что эта сущность является членом класса *Пицца*. Также известно, что сущность имеет, по крайней мере, одну начинку вида *СырНачинка*. Но эти знания не позволяют нам утверждать, что эта сущность принадлежит классу *СырПицца*. Чтобы сделать это возможным, мы должны задать не только необходимые, но и достаточные условия принадлежности к этому классу.

Класс, который имеет, по крайней мере, одно утверждение о необходимых и достаточных условиях, называется *определяемым классом*.

Для того чтобы конвертировать необходимые условия в необходимые и достаточные условия, они должны переместиться из-под заголовка «*Суперклассы*» под заголовок «*Эквивалентные классы*». Это можно сделать, используя опцию «*Конвертировать в определяемый класс (Convert to defined class)*» в «*Edit*» меню.



**Рис. 1.53.** Описание *СырПицца*  
(Использование необходимых и достаточных условий)

**Задание 5:** преобразуйте необходимые условия для *СырПицца* в необходимые и достаточные условия.

1. Убедитесь, что элемент *СырПицца* выбран в иерархии классов.

2. В меню «Edit» выберите пункт «Конвертировать в определяемый класс». Панель «Описание класса» приобретет вид, показанный на рис. 1.53.

Описанию класса в разделе «Эквивалентные классы» соответствует логическая формула *СырПицца* ≡ *Пицца and имеетНачинку some СырНачинка*. В этой формуле класс *СырПицца* эквивалентен анонимному классу *Пицца and имеетНачинку some СырНачинка*, ≡ – знак операции эквивалентности.

Подведем итог. *Необходимые и достаточные условия* для некоторого класса А означают следующее. Элементы этого класса обязательно удовлетворяют всем заданным условиям, но и если есть сущность, которая удовлетворяет заданным условиям, то она обязательно принадлежит классу А.

Чем это полезно на практике? Пусть имеется класс Б и известно, что любые сущности, которые являются членами класса Б, также удовлетворяют условиям, которые определяют класс А. Можно сделать вывод, что класс Б является подклассом А. Проверка правильности категоризации является одной из ключевых задач логики рассуждений, и можно использовать резонер для автоматического вычисления классификационной иерархии.

В OWL можно иметь несколько наборов необходимых и достаточных условий.

## Примитивные и определяемые классы

Классы, у которых есть хотя бы одно необходимое и достаточное условие, называются *определяемыми* – у них есть определение. Любая сущность, удовлетворяющая определению, будет принадлежать классу. Классы, у которых нет набора необходимых и достаточных условий (только необходимые условия), называются *примитивными* классами. В *Prot'eg'e 4* определяемые классы обозначаются значком с тремя горизонтальными белыми линиями в них. Для обозначения примитивных классов используется значок, который имеет простой коричневый фон. Важно также понимать, что резонер может только автоматически строить иерархию определяемых классов – то есть классов, по крайней мере, с одним набором необходимых и достаточных условий.

## Автоматическая классификация

Возможность использования рассуждений для автоматического вычисления иерархии классов является одним из основных преимуществ онтологии с использованием языка OWL. Действительно, при создании очень больших онтологий (свыше нескольких тысяч классов) использование рассуждений для вычисления отношения подкласс – суперкласс между классами имеет большое значение. Без автоматического вывода очень трудно управлять большими онтологиями и проверять их логическую корректность.

В случае множественного наследования, когда у класса несколько предков, удобно преобразовать такой вид наследования в простую иерархию (этот процесс называется нормализацией онтологии). Классы в иерархии, которые построены вручную, имеют не более одного суперкласса. Вычислить и управлять множественным наследованием – это работа резонера, именно он проводит нормализацию онтологии. Такая техника помогает сохранить онтологию управляемой и удовлетворяющей принципу модульности. Нормализация сводит к минимуму человеческие ошибки, которые возникают при создании множественного наследования, и упрощает процесс объединения с другими онтологиями.

Создав определение класса *СырПицца*, мы можем использовать резонер для автоматического вычисления подклассов класса *СырПицца*.

**Задание 6:** используйте резонер для автоматического вычисления подклассов класса *СырПицца*.

1. Нажмите кнопку «Старт резонер ...» в выпадающем меню пункта *Reasoner* главного меню (см. рис. 1.45). Через несколько секунд выводимая иерархия должна быть вычислена и появится в окне выводимой иерархии. Выводимая иерархия должна выглядеть примерно так, как показано на рис. 1.54.

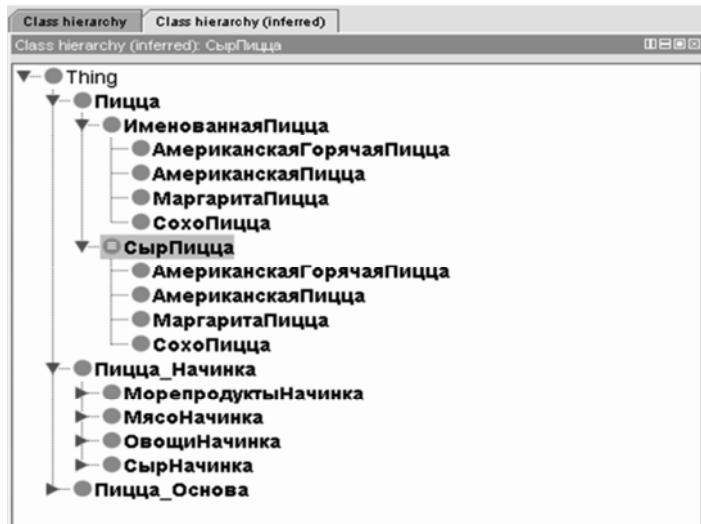


Рис. 1.54. Автоматическое вычисление подклассов класса *СырПицца*

Резонер определил, что *МаргаритаПицца*, *АмериканскаяПицца*, *АмериканскаяОстраяПицца* и *СохоПицца* являются подклассами класса *СырПицца*. Это потому, что мы определили *СырПицца* с использованием необходимых и достаточных условий. Любая пицца, если она имеет хотя бы одну начинку типа *СырНачинка*, является членом класса *СырПицца*. В связи с тем, что все сущности, которые принадлежат классам *МаргаритаПицца*, *АмериканскаяПицца*, *АмериканскаяОстраяПицца* и *СохоПицца*, являются элементами класса *Пицца* и у них есть, по крайней мере, одна начинка типа *СырНачинка*, резонер определил, что эти классы должны быть подклассами *СырПицца*. Важно понимать, что резонер никогда не создаст подклассов для примитивных классов.

## Универсальные ограничения

Все ограничения, которые мы создали до сих пор, были экзистенциальные ограничения (некоторые). Экзистенциальные ограничения указывают на *существование*, по крайней мере, одного элемента, принадлежащего указанному справа от свойства классу, участвующему в связи (отношении). Экзистенциальные ограничения не утверждают, что все элементы, участвующие в связи, являются элементами указанного справа класса. Чтобы указать, что связь объектов возможна только с элементами заданного класса, необходимо использовать *универсальные ограничения*.

*Универсальные ограничения* соответствуют квантору *всеобщности*. Они позволяют участвовать в связи для данного свойства только элементам заданного справа класса. Например, универсальное ограничение «имеетНачинку **only** МоцареллаНачинка» описывает пиццы, у которых все (любые) начинки относятся к классу *МоцареллаНачинка* и не может быть начинок, не являющихся элементами класса *МоцареллаНачинка*. Универсальные ограничения также известны как *AllValuesFrom* ограничения в языке OWL.

Описанному выше универсальному ограничению «имеетНачинку **only** МоцареллаНачинка» также соответствуют сущности, которые не участвуют ни в одном экземпляре отношения *имеетНачинку*, т.е. пиццы, не имеющие начинок.

Предположим, мы хотим создать класс *ВегетарианецПицца*. Все пиццы этого класса должны иметь только начинки, которые относятся к типу *СырНачинка* или *ОвощиНачинка*. Для этого мы можем использовать универсальные ограничение.

**Задание 7:** создайте класс для описания *ВегетарианецПицца*.

1. Создайте подкласс класса Пицца и назовите его *ВегетарианецПицца*.

2. Убедитесь, что выделен класс *ВегетарианецПицца*, и нажмите на «Добавить» – значок (+) рядом с заголовком «*Суперкласс*» на панели «*Описание класса*».

3. Наберите *имеетНачинку* как свойство для ограничения.

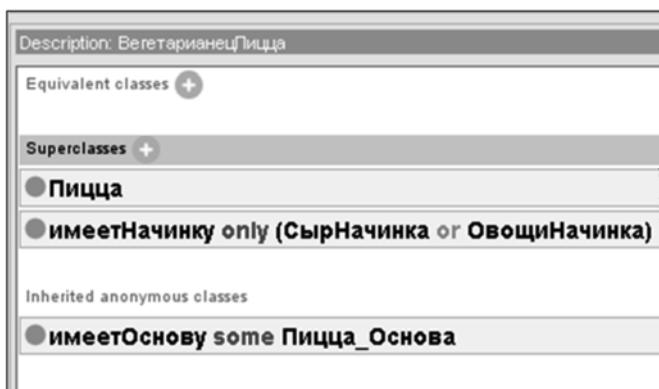
4. Наберите **only**, чтобы создать универсальное ограничение.

5. Для правого аргумента отношения мы хотим сказать *СырНачинка* или *ОвощиНачинка*. Это можно записать, используя оператор объединения (**or**), как выражение в скобках «(*СырНачинка or ОвощиНачинка*)».

6. Нажмите ОК, чтобы закрыть диалоговое окно и создать ограничение. В этот момент панель «*Описание класса*» будет иметь вид, показанный на рис. 1.55.

Это означает, что если сущность является членом класса *ВегетарианецПицца*, то она обладает следующими свойствами: является элементом класса *Пицца*, имеет в качестве начинки только начинки типа *СырНачинка* либо типа *ОвощиНачинка*.

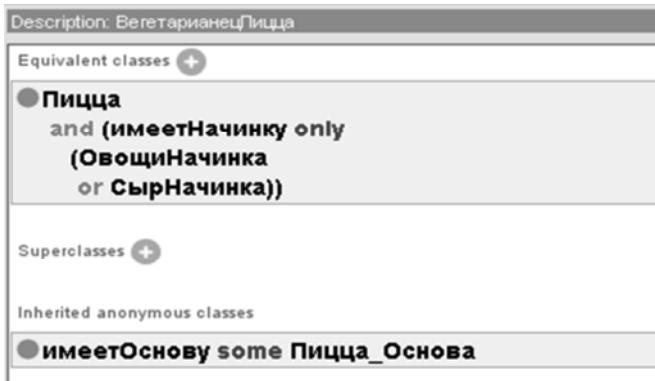
В ситуациях, как в примере выше, распространенной ошибкой является использование пересечения (*and*) вместо объединения (*or*). Например, *СырНачинка and ОвощиНачинка*, что читается как «*СырНачинка и ОвощиНачинка*». Хотя «*СырНачинка и ОвощиНачинка*» может звучать естественно на русском языке, с логической точки зрения это означает, что начинка одновременно относится к типу *СырНачинка* и к типу *ОвощиНачинка*. Это явно неверно.



**Рис. 1.55.** Описание *ВегетарианецПицца*  
(использование необходимых условий)

В приведенном выше примере было бы заманчиво создать два универсальных ограничения – одно для *СырНачинка* (*имеетНачинку only СырНачинка*) и одно для *ОвощиНачинка* (*имеетНачинку only ОвощиНачинка*). Тем не менее, когда несколько ограничений используется, общее описание получается путем выполнения операции пересечения отдельных ограничений. Два таких ограничения эквивалентны одному ограничению с наполнителем, полученным путем пересечения множеств начинок типа *МоцареллаНачинка* и *ОвощиНачинка*, – как описано выше, это было бы логически неверно.

Так как любая сущность, удовлетворяющая двум рассмотренным условиям, должна быть элементом *ВегетарианецПицца*, то следует конвертировать необходимые условия для *ВегетарианецПицца* на *необходимые и достаточные условия*. Это также позволит использовать резонер, чтобы определить подклассы класса *ВегетарианецПицца*.



**Рис. 1.56.** Определение класса *ВегетарианецПицца*  
(используя необходимые и достаточные условия)

**Задание 8:** преобразуйте необходимые условия для *ВегетарианецПицца* в необходимые и достаточные условия. Панель «Описание класса» должна иметь вид, показанный на рис. 1.56. Мы превратили наше описание *ВегетарианецПицца* в определение.

### Рассуждения в открытом мире

Мы хотим использовать резонер для автоматического вычисления отношений суперкласс – подкласс (отношений категоризации) между *МаргаритаПицца* и *ВегетарианецПицца*, а также – *СохоПицца* и *ВегетарианецПицца*. Напомним, что мы считаем, что *МаргаритаПицца* и *СохоПицца* должны быть вегетарианской пиццей (они должны быть подклассами *ВегетарианецПицца*). Имея ранее созданное определение *ВегетарианецПицца*, можно использовать резонер для выполнения автоматической классификации и определить, что такое вегетарианская пицца в нашей онтологии.

**Задание 9:** используйте резонер для классификации онтологии.

1. Нажмите кнопку «Старт резонер...» в раскрывающемся меню резонера. Вы заметите, что *МаргаритаПицца*, а также *СохоПицца* не были классифицированы как подклассы *ВегетарианецПицца*.

Это может показаться немного странным, так как создается впечатление, что обе *МаргаритаПицца* и *СохоПицца* имеют вегетарианские ингредиенты, то есть компоненты, которые относятся к типу *СырНачинка* или типу *ОвощиНачинка*. Однако, как мы увидим, кое-что отсутствует в определении *МаргаритыПиццы* и *СохоПиццы*, в результате они не могут быть классифицированы как подклассы *ВегетарианецПицца*.

Рассуждения OWL (описание логики) основаны на так называемом предположении об открытости мира (OWA). Их часто называют рассуждениями в открытом мире (OWR). Предположение об открытости мира означает, что мы не можем утверждать, что сущность или отношение не существует, если это не указано явно. Другими словами, если о каком-то отношении не сказано, что оно истинно, то нельзя считать его ложным. Предполагается, что «знания просто не были добавлены в базу знаний». В случае нашей онтологии пиццы мы заявили, что *МаргаритаПицца* имеет начинки типа *МоцареллаНачинка*, а также типа *ПомидорыНачинка*. Из-за предположения об открытости мира пока мы явно не сказали, что *МаргаритаПицца* имеет только эти виды начинок, будет предполагаться (по рассуждениям), что у *МаргаритыПиццы* могут быть и другие начинки. Чтобы явно указать, что *МаргаритаПицца* имеет начинки, которые являются видами *МоцареллаНачинка* или видами *ПомидорыНачинка* и только видами *МоцареллаНачинка* или *ПомидорыНачинка*, мы должны добавить так называемую аксиому закрытия о свойстве *имеетНачинку*.

### **Аксиома закрытия**

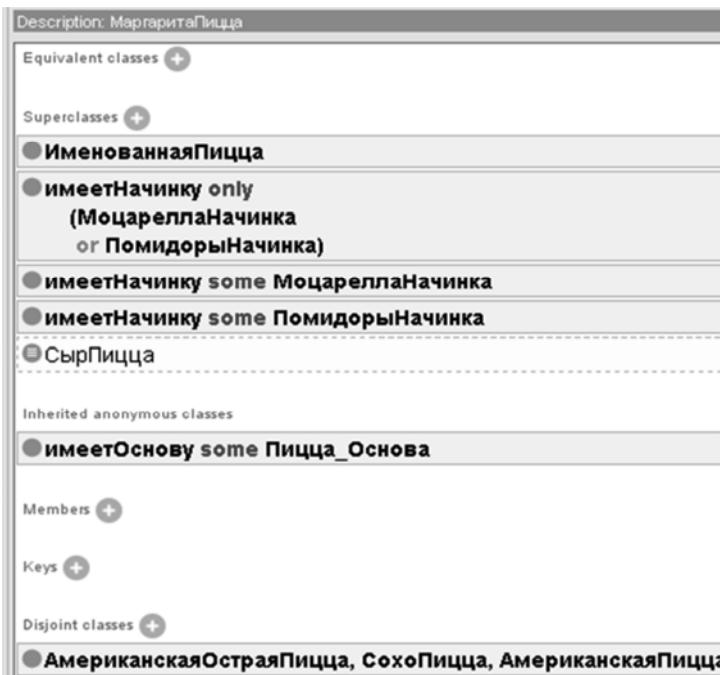
Аксиома закрытия для свойства состоит из универсальных ограничений, которые указывают, что пицца может быть только заполнена указанными наполнителями. Например, аксиома закрытия для *МаргаритаПицца* у свойства *имеетНачинку* является универсальным ограничением, которое описывает наполнитель пиццы как объедине-

ние *МоцареллаНачинка*, а также *ПомидорыНачинка* и имеет вид «имеетНачинку **only** (*МоцареллаНачинка or Помидоры Начинка*)».

**Задание 10:** добавьте аксиому закрытия к свойству *имеетНачинку* для класса *МаргаритаПицца*.

1. Убедитесь, что класс *МаргаритаПицца* выбран в иерархии классов. Нажмите «Добавить» – значок (+) рядом с разделом «Суперкласс», чтобы открыть окно редактирования выражений.

2. Наберите *имеетНачинку only* (*МоцареллаНачинка or ПомидорыНачинка*). Нажмите ОК для создания ограничения. Панель «Описание класса ...» будет иметь вид, как показано на рис. 1.57.



**Рис. 1.57.** Панель «*Описание класса ...*» для *МаргаритаПиццы* с использованием аксиомы закрытия

Распространенной ошибкой является использование только универсальных ограничений в описании. Например, попробуем описать класс *МаргаритаПицца* как подкласс класса *Пицца* и зададим только универсальное ограничение «имеетНачинку **only** (*МоцареллаНачинка or Помидоры Начинка*)» без указания экзис-

тенциальных ограничений. Из-за семантики универсальных ограничений полученное описание означает, что элемент *МаргаритаПицца* – это либо пицца, имеющая только начинки типа *МоцареллаНачинка* или *ПомидорыНачинка*, либо сущность, которая является **пиццей и не имеет никаких начинок вообще**.

Аналогично создайте аксиомы закрытия для классов *СохоПицца*, *АмериканскаяОстраяПицца*.

Добавив аксиомы закрытия (замыкания) к свойству *имеетНачинку* для нашей пиццы, мы можем использовать резонер (мыслитель) для автоматического вычисления классификации. Нажмите кнопку «Старт резонер ...» в выпадающем меню для вызова Резонера (*Машины вывода*). После короткого периода времени онтология будет классифицирована и появится на панели «*Выводимая иерархия*».

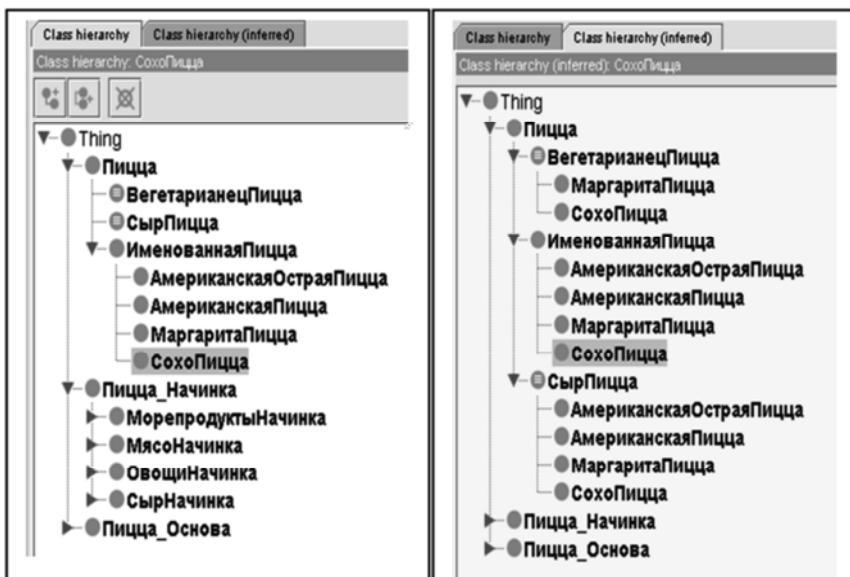


Рис. 1.58. Присоединенная и выводимая иерархии после классификации с помощью резонера

На этот раз *МаргаритаПицца*, а также *СохоПицца* будут классифицированы как подклассы *ВегетарианецПицца*. Это произошло потому, что мы специально «закрыли» свойство *имеетНачинку*, чтобы точно сказать, что *ВегетарианецПицца* определяется как

Пицца с начинками только вида *СырНачинка* и только вида *ОвощиНачинка*. Рис. 1.58 показывает *присоединенную и выводимую иерархию*. Ясно видно, что присоединенная иерархия проще и «прозрачнее», чем выводимая иерархия. Однако использование рассуждений помогает (особенно в случае больших онтологий) при поддержании нескольких иерархий наследования.

### **Контрольные вопросы**

1. Для каких целей используется резонер (машина вывода)?
2. Что такое присоединенная иерархия и выводимая иерархия?
3. Как проявляется противоречивый класс?
4. Как вы понимаете необходимые и достаточные условия? Как их описать?
5. Что такое примитивный и определяемый класс?
6. Как преобразовать необходимые условия в необходимые и достаточные условия?
7. Что такое универсальные ограничения? Как их задать?
8. Как понимать автоматические рассуждения в открытом мире?
9. Что такое аксиома закрытия? Как ее создать?

## **1.5. Построение иерархии классов**

### **Списки значений**

Каждый класс может быть разбит на подгруппы разными способами в зависимости от выбранного для классификации признака или свойства объекта. Для логического описания процесса разбиения объектов на группы по некоторому признаку необходимо определить списки возможных значений этого признака.

Списки значений являются шаблонами проектирования. Шаблоны проектирования в дизайне онтологии, аналогичные шаблоны проектирования в объектно-ориентированном программировании являются решениями для моделирования проблемы, которая встречается многократно. Эти шаблоны проектирования разрабатываются экспертами и в настоящее время признаются как проверенный продукт для решения общих проблем моделирования. Как упоминалось ранее, списки значений должны быть созданы для того, чтобы выполнить описание класса на языке логики. Например, создадим список значений, который назовем «*СписокЗначенийПря-*

ности», чтобы описать «пряности» для начинок класса *Пицца\_Начинка*. Списки значений ограничивают круг возможных значений некоторой величины, задавая их исчерпывающий список. Например, наш *«СписокЗначенийПряности»* будет иметь следующий диапазон значений: *«Мягкая»*, *«Средняя»* и *«Острая»*. Создание списка возможных значений состоит из нескольких этапов.

1. Создание класса для представления списка значений. Например, чтобы получить список значений пряности, нужно создать класс *СписокЗначенийПряности*.

2. Создание подклассов списка значений, чтобы представить его возможные варианты. Например, можно создать классы мягких, средних и острых начинок как подклассы класса *СписокЗначенийПряности*.

3. Определение подклассов непересекающимися.

4. Добавление аксиомы покрытия, чтобы сделать список значений исчерпывающим (см. следующий раздел).

5. Создание свойства (отношения, связи) для объектов класса *Пицца\_Начинка* и объектов класса *СписокЗначенийПряности*. Например, можно создать свойство *имеетПряность*.

6. Определение свойства функциональным.

7. Определение диапазона свойства как класса *СписокЗначенийПряности*.

Давайте создадим списки значений, которые могут быть использованы для описания пряности начинки пиццы. Благодаря этим спискам можно будет классифицировать нашу пиццу по степени пряности, потому что начинки пиццы могут быть по пряности либо «мягкими», либо «средними», либо «острыми».

Обратите внимание, что указанные элементы взаимно исключают друг друга – сущность не может быть и «мягкой» и «острой» одновременно или быть сочетанием имеющихся вариантов.

**Задание 1.** Создайте список значений для описания пряности начинки пиццы.

1. Создайте новый класс как подкласс *Thing*, и назовите его *СпискиЗначений*.

2. Создайте подкласс класса *СпискиЗначений* и назовите его *СписокЗначенийПряности*.

3. Создайте три новых класса как подклассы *СписокЗначенийПряности*. Имена этих классов *«Острая»*, *«Средняя»* и *«Мягкая»*.

4. Сделайте классы «*Острая*», «*Средняя*» и «*Мягкая*» не пересекающимися друг с другом. Вы можете сделать это, выбрав класс «*Острая*» и выбрав пункт «*Сделать все примитивные классы одного уровня непересекающимися*» из пункта «*Edit*» в меню.

5. На вкладке «*Свойства объектов (Object Property)*» создайте новое свойство и назовите его *имеетПряность*. Установить диапазон этого свойства *СписокЗначенийПряности*. Сделайте это новое свойство функциональным.

6. Добавьте *аксиому покрытия* к классу *СписокЗначенийПряности*. Выделите элемент *СписокЗначенийПряности* в иерархии классов. В разделе «*Эквивалентные классы*» выберите «*Добавить*» значок (+) и в текстовом окне введите «*Острая or Средняя or Мягкая*». Давайте посмотрим на класс *СписокЗначенийПряности* (рис. 1.59 и рис. 1.60).

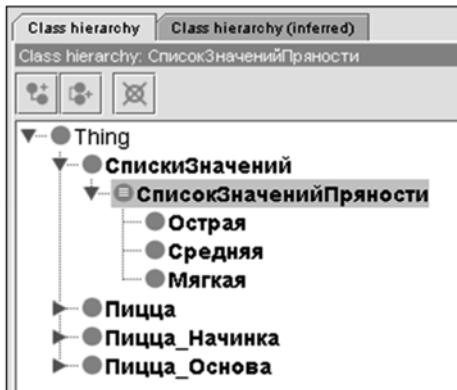


Рис. 1.59. Создание класса *СписокЗначенийПряности*

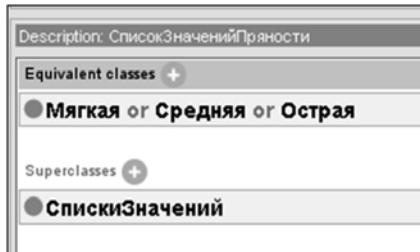


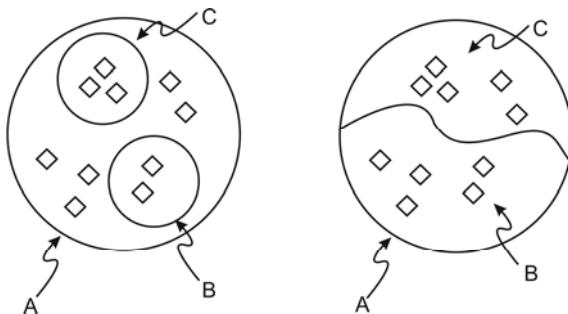
Рис. 1.60. Описание класса *СписокЗначенийПряности*

## Аксиома покрытия

Аксиома покрытия состоит из двух частей: класс, который в настоящее время «покрыт», и классы, которые формируют покрытие. Например, пусть у нас есть три класса *A*, *B* и *C*. Классы *B* и *C* являются подклассами класса *A*. Теперь предположим, что у нас есть аксиома покрытия, которая определяет, что класс *A* покрыт классом *B*, а также классом *C*. Это означает, что член класса *A* должен быть членом группы *B* и/или *C*. Если классы *B* и *C* не пересекаются, то член класса *A* должен быть членом либо класса *B*, либо класса *C*.

Таким образом, аксиома покрытия описывает класс, являющийся объединением непересекающихся классов.

Помните, что обычно, хотя *B* и *C* являются подклассами *A*, существует сущность, которая может быть членом класса *A*, не будучи членом *B* или *C*. Аксиома покрытия изображена на рис. 1.61.



**Рис. 1.61.** Схема, которая показывает использование аксиомы покрытия для покрытия класса *A* классами *B* и *C*

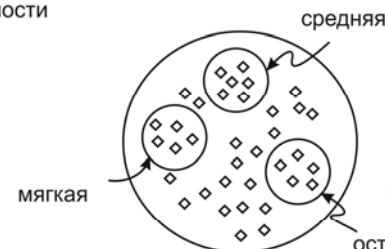
Пусть *СписокЗначенийПряности* имеет аксиому покрытия, которая утверждает, что класс *СписокЗначенийПряности* покрыт классами *Мягкая*, *Средняя* и *Острая*. Классы *Мягкая*, *Средняя* и *Острая* не пересекаются друг с другом, так что сущность не может быть членом более чем одного из них. Класс *СписокЗначенийПряности* имеет суперкласс, который описывается как объединение классов «*Мягкая or Средняя or Острая*». Различие между случаями использования и неиспользования аксиомы покрытия иллюстрируется рис. 1.62.

В обоих случаях классы *Мягкая*, *Средняя* и *Острая* не пересекаются. Видно, что в случае без аксиомы покрытия сущность может быть членом класса *СписокЗначенийПряности* и не быть членом одного из классов *Мягкая*, *Средняя* или *Острая*. *СписокЗначенийПряности* не покрывается классами *Мягкая*, *Средняя* и *Острая*. Сравните это со случаем, когда применяется аксиома покрытия. Видно, что если сущность является членом класса *СписокЗначенийПряности*, она должна быть членом одного из трех подклассов *Мягкая*, *Средняя* или *Острая*. *СписокЗначенийПряности* покрыт классами *Мягкая*, *Средняя* и *Острая*.

### Добавление свойства пряности

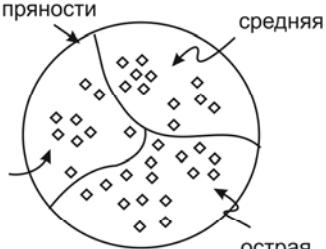
Теперь можно использовать *СписокЗначенийПряности*, чтобы описать пряность начинок нашей пиццы. Для этого добавим экзистенциальное ограничение для каждого вида *Пицца\_Начинка* по свойству «пряность». Ограничение будет иметь вид: «имеетПряность *some* СписокЗначенийПряности», где *СписокЗначенийПряности* будет одним из классов *Мягкая*, *Средняя* или *Острая*. Это можно сделать для каждого вида начинки, как ранее делалось для различных видов пиц.

Классификация значений пряности



Без аксиомы покрытия

Классификация значений пряности



С аксиомой покрытия (все значения пряности покрыты подклассами: острая, мягкая и средняя)

**Рис. 1.62.** Использование аксиомы покрытия для класса *СписокЗначенийПряности*

**Задание 2.** Создайте ограничения по свойству *имеетПряность* для начинок пиццы (*Пицца\_Начинка*).

1. Убедитесь, что *Джалапено\_перецНачинка* выбран в иерархии классов.

2. Используйте «Добавить» значок (+) в разделе «Суперкласс» панели «Описание класса». Откроется диалоговое окно.
3. Выберите вкладку «Редактор выражений для классов».
4. Наберите «имеетПряность some Острая» для создания экзистенциального ограничения. Помните, что можно использовать автозаполнение для ускорения процесса.
5. Нажмите 'OK', чтобы создать ограничение, – если есть любые ошибки, ограничения не будут созданы и ошибки будут выделены ярким цветом.

6. Повторите это для каждого вида начинки того же уровня.

Чтобы завершить этот раздел, создадим новый класс *ПрянаяПицца*, т.е. пицца с пряными начинками. Для того чтобы сделать это, определим класс *ПрянаяПицца* как *Пицца*, имеющая хотя бы одну начинку с пряностью (*имеетПряность*) типа *Острая*.

**Задание 3.** Создайте класс *ПрянаяПицца* как подкласс класса *Пицца*.

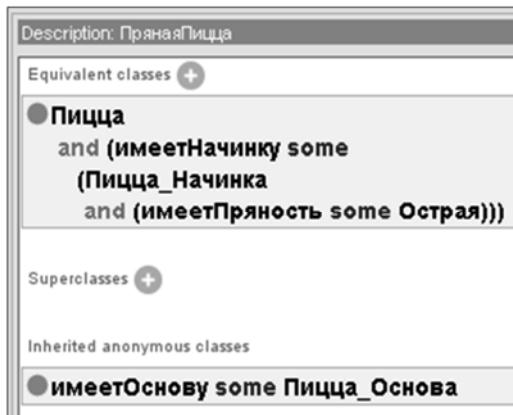
1. Создайте подкласс класса *Пицца* и назовите его *ПрянаяПицца*.
2. Убедитесь, что выделен класс *ПрянаяПицца*.
3. Нажмите «Добавить» значок (+) в разделе «Суперкласс» и в редакторе выражений наберите выражение «имеетНачинку some (*Пицца\_Начинка and имеетПряность some Острая*)».

Выражение «*Пицца\_Начинка and имеетПряность some Острая*» описывает анонимный класс, который содержит члены класса *Пицца\_Начинка*, имеющие хотя бы одну связь с элементами класса *Острая*.

4. В завершение выберите опцию «Преобразовать в определяемый класс» в пункте «Edit» главного меню. Описание класса *ПрянаяПицца* будет иметь вид, показанный на рис. 1.63.

Описание класса *ПрянаяПицца* говорит, что все его члены являются элементами класса *Пицца* и имеют хотя бы одну начинку, пряность которой относится к типу *Острая*. Оно также говорит, что любая пицца, если она имеет хотя бы одну начинку, которая имеет пряность типа *Острая*, является элементом класса *ПрянаяПицца*. Теперь, используя рассуждения, можно определить все пряные пиццы в онтологии.

Запустите на выполнение резонер для классификации онтологии. Выберите в выпадающем меню резонера пункт «*Synchronize resoner...*» для классификации онтологии.



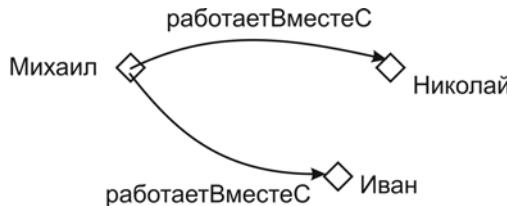
**Рис. 1.63.** Определение класса *ПрянаяПицца*

Как только процесс вывода завершится, откройте вкладку «*Выvodимая иерархия*». Вы должны увидеть, что *АмериканскаяОстраяПицца* была классифицирована как подкласс класса *ПрянаяПицца* – резонер автоматически определил, что любой член класса *АмериканскаяОстраяПицца* является также членом класса *ПрянаяПицца*.

### Ограничения на мощность связей

Мощность связей – это количество экземпляров отношения  $P$ , присутствующее в предметной области для одного элемента описываемого класса.

В OWL можно описать категорию объектов, которые имеют, по крайней мере, ровно указанное количество связей с другими объектами или данными определенного типа. Ограничения, которые описывают подобные классы, известны как ограничения мощности (*Cardinality Restrictions*). Для данного свойства  $P$  минимальное ограничение мощности (*Minimum Cardinality Restriction*) указывает минимальное количество экземпляров отношения  $P$ , которые должны присутствовать в предметной области для одного объекта. Максимальное ограничение мощности (*Maximum Cardinality Restriction*) определяет максимальное число экземпляров отношения  $P$ , и просто ограничение мощности (*Cardinality Restriction*) указывает точное число экземпляров отношения  $P$ , которые должны присутствовать в предметной области для одного объекта.



**Рис. 1.64.** Ограничение мощности указывает число экземпляров связи

Например, на рис. 1.64 изображены индивидуум *Михаил* и связанные с ним через свойство *работаетВместеС* индивидуальности *Николай* и *Иван*. Индивидуум *Михаил* удовлетворяет минимальному ограничению по мощности 2 для свойства *работаетВместеС*, если личности *Николай* и *Иван* отличаются друг от друга.

Давайте добавим ограничение мощности в нашу онтологию пиццы. Создадим новый подкласс класса *Пицца*, который назовем *ИнтереснаяПицца*, у нее должно быть три или более начинки.

**Задание 4.** Создайте класс *ИнтереснаяПицца*, у элементов которого не менее трех начинок.

1. Переключитесь на вкладку *Классы* и убедитесь, что выбран класс *Пицца*.

2. Создайте подкласс класса *Пицца* и назовите его *ИнтереснаяПицца*.

3. Нажмите «Добавить» значок (+) в разделе «*Суперкласс*».

4. В диалоговом редакторе выражений наберите ограничение: «имеетНачинку *min 3*».

5. Нажмите «*Enter*», чтобы закрыть диалоговое окно и создать ограничение.

В панели описания класса в разделе «*Суперкласс*» вы увидите «*Пицца and имеетНачинку min 3*». Раздел «*Эквивалентные классы*» должен быть пустым.

6. Выберите опцию «*Преобразовать в определяемый класс*» в меню «*Edit*».

Секция «*Суперкласс*» теперь пуста, в то время как в секции «*Эквивалентные классы*» появилось ограничение: «*Пицца and имеетНачинку min 3*». Описание класса *ИнтереснаяПицца* представлено на рис. 1.65.

Что это значит? Определение *ИнтереснаяПицца* описывает множество объектов (пицц), которые являются членами класса

*Пицца* и имеют, по крайней мере, три начинки, т.е. отношение *имеетНачинку* связывает каждый из элементов класса *Интересная Пицца* не менее чем с тремя элементами из класса *Пицца\_Начинка*.

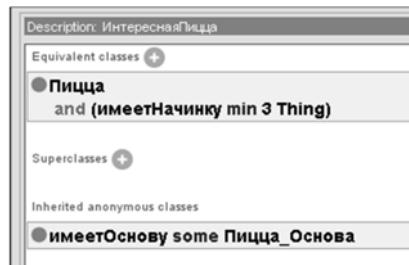


Рис. 1.65. Панель «Описание класса *ИнтереснаяПицца*» с числом начинок не менее трех

Используйте резонер, чтобы классифицировать онтологию. Нажмите «Синхронизировать резонер ...» в раскрывающемся меню пункта *Резонер*. На вкладке «Выводимая иерархия» вы увидите картину, показанную на рис. 1.66.

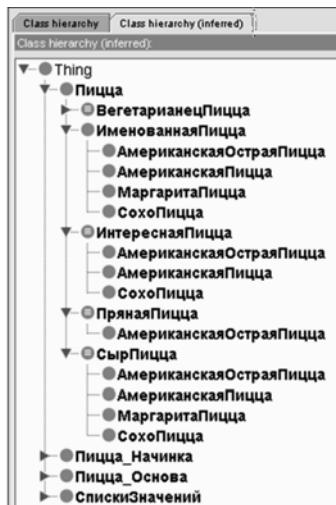


Рис. 1.66. Класс *ИнтереснаяПицца* содержит пиццы с числом начинок не менее трех

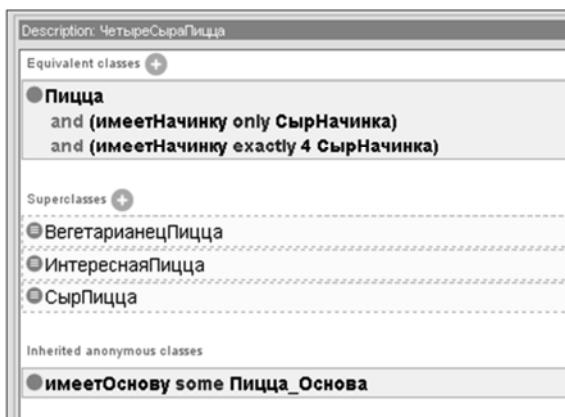
Обратите внимание, что *ИнтереснаяПицца* теперь имеет подклассы – *АмериканскаяПицца*, *АмериканскаяОстраяПицца* и *Сохо*

*Пицца*, однако *МаргаритаПицца* не отнесена к *ИнтереснойПицце*, потому что она имеет только два различных вида начинки.

### Ограничения с заданным значением мощности

В предыдущем разделе описаны ограничения мощности, когда указывается минимальное (или максимальное) число отношения  $P$ , в которых участвует индивид. В этом разделе определим конкретные значения ограничений мощности.

Давайте добавим ограничение с заданным значением мощности к нашей онтологии. Чтобы сделать это, создадим подкласс класса *ИменованнаяПицца* и назовем его *ЧетыреСырапицца*, элементы которого должны иметь ровно четыре сырных начинки.



**Рис. 1.67.** Описание *ЧетыреСырапицца* с использованием ограничений мощности

**Задание 5.** Создайте класс *ЧетыреСырапицца*, элементы которого имеют ровно четыре сырных начинки (рис. 1.67).

1. Создайте подкласс класса *Пицца* и назовите его *ЧетыреСырапицца*.
2. Выберите заголовок «*Суперкласс*» в панели «*Описание класса ...*» и нажмите «*Добавить*» значок (+), чтобы открыть редактор выражений.
3. Наберите в редакторе выражение «*имеетНачинку exactly 4 СырНачинка*»
4. Нажмите *OK* и создайте ограничение.

Наше определение класса *ЧетыреСыраПицца* описывает множество индивидов, входящих в класс с именем *Пицца* и имеющих ровно четыре экземпляра отношения *имеетНачинку* с элементами класса *СырНачинка*. При таком описании *ЧетыреСыраПицца* все еще может иметь связи и с другими видами начинок. Для того чтобы сказать, что мы хотим, чтобы имелось только четыре сырных начинки и других начинок не могло быть, мы должны добавить ключевое слово *only* (квантор всеобщности). Это означает, что разрешены только виды сырных начинок. Описание класса должно иметь вид, показанный на рис. 1.67.

### **Контрольные вопросы**

1. Что такое аксиома покрытия?
2. Как создать аксиому покрытия?
3. Что такое списки значений некоторого свойства? Как их создать?
4. Что такое мощность связей? Какие ограничения на мощность связей вы знаете?
5. Как создать ограничение на мощность связей?
6. Что означает ограничение с заданным значением мощности связей? Каким ключевым словом оно задается?
7. Что означает минимальное или максимальное ограничение мощности? Какими ключевыми словами эти ограничения задаются?

## **1.6. Характеристики объектов (*DataProperties*)**

### **Связи объектов с типизированными характеристиками**

Свойства, рассмотренные в § 1.2–1.5, описывают отношения (связи) между объектами классов. Здесь будем рассматривать свойства (*Data Properties*), описывающие связь объектов с их характеристиками (атрибутами), значения которых задаются с помощью данных определенного типа. Тип данных свойства задается с помощью ссылки на типы данных *XML*-схемы или в виде значения *RDF*-литерала.

Другими словами, *Data Properties* описывают отношения между индивидом и значением данных, представляющих характеристики индивида. Свойства данных (*Data Properties*) могут быть

созданы с помощью инструментов, расположенных на вкладке *Data Properties*, показанной на рис. 1.68.

Будем использовать *Свойства данных* (*Data Properties*) для описания калорийности пиццы. При этом введем некоторые числовые диапазоны, чтобы классифицировать виды пиццы, такие как высококалорийная или низкокалорийная пицца.

Для этого нам нужно выполнить следующие шаги.

– Создать свойство данных (*Data Properties*) *имеетКалорийность*, которое будет устанавливать значение калорийности определенной пиццы.

– Создать несколько примеров пицц с конкретным содержанием калорий.

– Создать два класса, разделяющих пиццы на две категории: низкокалорийная пицца и высококалорийная пицца.

Теперь давайте сделаем это в *Prot'eg'e*. Свойства данных (*Data Properties*) могут использоваться, чтобы связать индивидуальности со значениями их характеристик, относящимися к определенному типу данных, например, к типу *integer* или *string*.

**Задание 1.** Создайте свойство данных *имеетКалорийность*.

1. Переключитесь на вкладку «*Data Properties*».

2. Используйте кнопку «Добавить *Data Properties*», чтобы создать новое свойство данных и назовите его *имеетКалорийность*.

Создадим несколько экземпляров пицц и укажем для них присущее им число калорий.

**Задание 2.** Создайте примеры конкретных пицц (индивидуальностей).

1. Убедитесь, что выбрана вкладка «*Индивидуальности*» или вкладка «*Сущности (Объекты)*», в нижней части которой надо переключиться на вкладку «*Индивидуальности выделенного типа*».

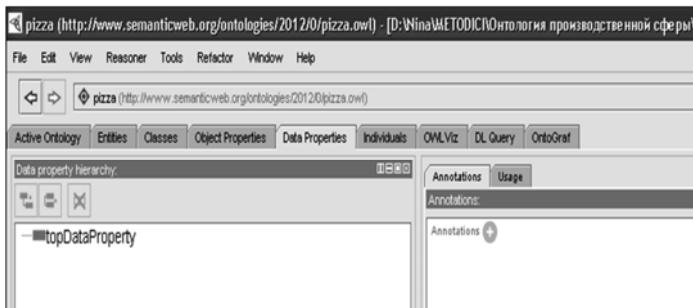
2. Нажмите кнопку «Добавить индивидуум» и введите его имя – *ПримерМаргариты*.

3. В панели обзора свойств индивидуальности укажите тип – *МаргаритаПицца*. Вы это можете сделать в появившемся диалоговом окне либо путем выбора элемента дерева иерархии классов, либо набрав имя в редакторе выражений с классами.

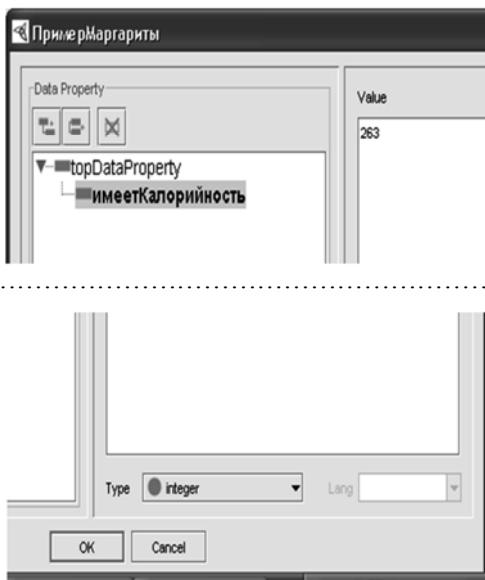
4. В панели «*Определение связей ...*» выберите секцию «*Определение связей с данными (Data Properties)*» и нажмите кнопку

«Добавить». В появившемся диалоговом окне, показанном на рис. 1.69, выберите в качестве свойства данных *имеетКалорийность*, *Integer(целое)* – в качестве типа данных и введите значение 263 в текстовом поле.

5. Создайте еще несколько примеров пиццы с различным содержанием калорий, в том числе экземпляр *ЧетыреСыра* с 723 калориями.



**Рис. 1.68.** Вкладка *Data Properties*, на которой создаются связи с типизированными данными



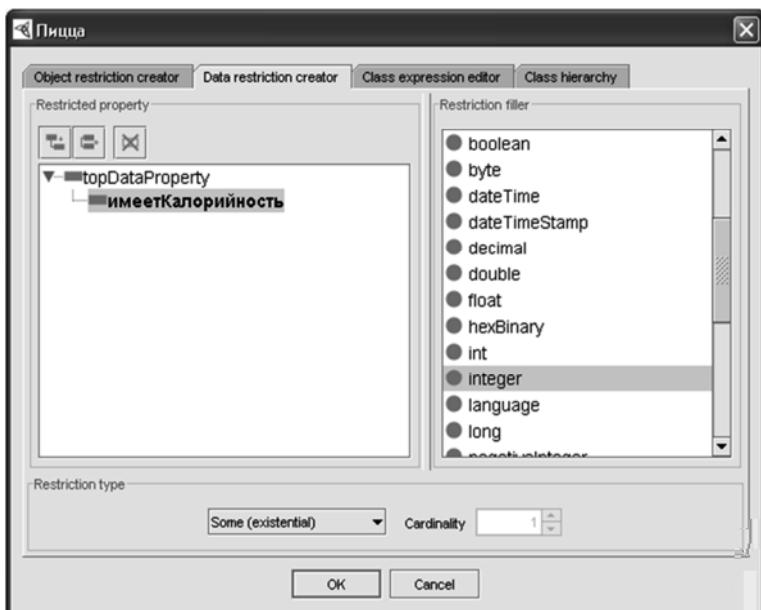
**Рис. 1.69.** Задание значения свойства данных для индивидуальности

Свойства данных могут использоваться для организации связей объектов с сущностями заданного типа. Встроенные типы данных описаны в словаре схемы *XML* и включают в себя целые числа, числа с плавающей точкой, строки, логические данные и т.д.

**Задание 3.** Создайте ограничения с использованием свойства данных, устанавливающего, что все *Пиццы* имеют калорийность.

1. Убедитесь, что выбран элемент *Пицца* на вкладке *Классы*.
2. Нажмите кнопку «Добавить» в секции «Суперкласс». Это приводит к появлению диалогового окна.
3. В диалоговом окне выберите вкладку «Создать ограничение с использованием свойств данных (Class restriction creator)», в левой части выберите свойство *имеетКалорийность* (рис. 1.70).
4. Установите тип ограничения *some*.
5. В правой части укажите тип данных *Integer* (*Целое число*).
6. Нажмите 'OK'. Ограничение «имеетКалорийность some integer» появилось в секции «Суперкласс».

С помощью этого ограничения мы объявили, что любая пицца имеет, по крайней мере, одно значение калорийности целого типа.

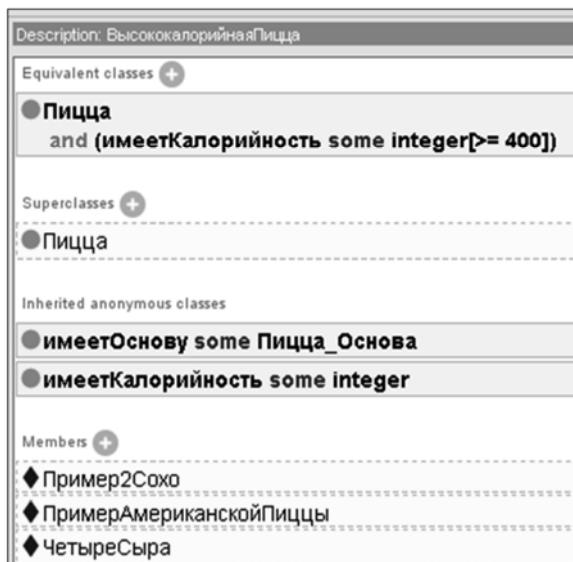


**Рис. 1.70.** Создание ограничения с использованием свойства данных

Используя типизированные данные, можно задавать ограничения на их значения. Например, можно определить диапазоны значений для характеристик, задаваемых числами, при этом могут использоваться такие понятия, как минимальное значение и максимальное значение. Как пример, *ВысококалорийнаяПицца* будет определена как пицца, у которой число калорий больше или равно 400.

**Задание 4.** Создайте класс *ВысококалорийнаяПицца*, у элементов которого калорийность выше или равна 400.

1. Переключитесь на вкладку «Классы».
2. Создайте подкласс класса *Пицца* и назовите его *ВысококалорийнаяПицца*.
3. На панели «Описание класса» в секции «Суперкласс» нажмите кнопку «Добавить» – значок (+), в редакторе выражений наберите «имеетКалорийность some integer [ $\geq 400$ ]» и нажмите кнопку «OK».
4. Конвертируйте примитивный класс в определяемый. Описание класса *ВысококалорийнаяПицца* будет иметь вид, показанный на рис. 1.71.



**Рис. 1.71.** Использование ограничений для описания диапазонов изменения калорийности *ВысококалорийнойПиццы*

5. Создайте класс *НизкокалорийнаяПицца* таким же способом, но определите ее следующим образом «*Пицца and имеетКалорийность some integer [<400]*» (любая пицца, калорийность которой меньше чем 400).

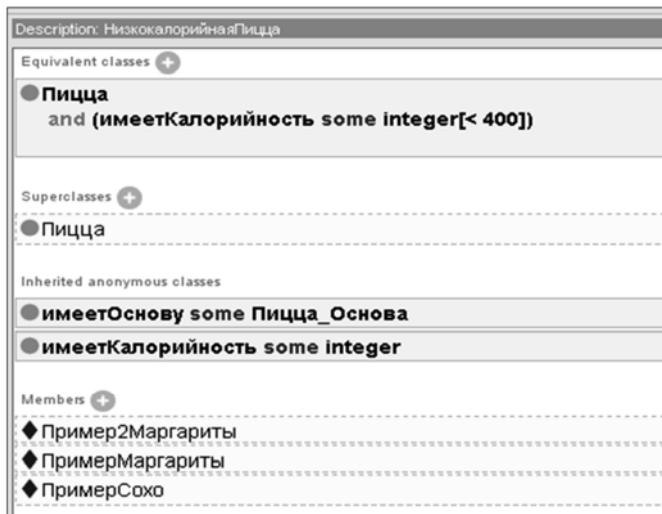
Обратите внимание, что описания различаются у *ВысококалорийнаяПицца* и у *НизкокалорийнаяПицца*.

Теперь у вас есть две категории пиццы, отличающиеся по калорийности. Проверим, осуществляется ли классификация для созданных нами примеров пицц, для которых мы указали различные значения калорий.

**Задание 5.** Классифицируйте пиццы на основе значений калорий (свойство *имеетКалорийность*).

1. Выберите опцию «Синхронизировать резонер» или «Старт резонер» в выпадающем меню пункта Резонер.

2. Выделите *ВысококалорийнаяПицца*. В правой части панели «Описание класса» серым цветом будут окрашены результаты, полученные путем вывода.



**Рис. 1.72.** Пиццы, отнесенные к категории *НизкокалорийнаяПицца* в результате работы резонера

3. Обратите внимание на секцию «Члены (Members)» (см. рис. 1.71). Она должна включать экземпляры «ЧетыреСыра» и, возможно, другие названия пицц, для которых вы указали калорийность 400 или больше.

4. Выберите *НизкокалорийнаяПицца*. Проверьте секцию «Члены (*Members*)». Она должна включать пиццу *«ПримерМаргариты»* и, возможно, другие пиццы, у которых значение калорий меньше чем 400 (рис. 1.72).

Проанализируем свойство *имеетКалорийность*. Любая пицца может иметь только одно значение калорийности, т.е. связь элемента класса *Пицца* со значением его калорий является функцией. Для отражения этого факта будем использовать характеристику *Data Properties (Свойства данных) Functional*. Для этого поставим галочку против этой характеристики. Протестируйте, что данное ограничение работает, создавая пиццы, у которых имеется два значения калорийности. Это должно приводить к непоследовательности онтологии.

### **Рассуждения в открытом мире**

Примеры этой главы демонстрируют нюансы рассуждений в открытом мире.

Создадим класс *НевегетарианскаяПицца*, чтобы добавить его к нашей классификации пиццы – к классу *ВегетарианскаяПицца*. Класс *НевегетарианскаяПицца* должен содержать все *Пиццы*, которые не принадлежат классу *ВегетарианскаяПицца*, потому является дополнением к нему.

**Задание 6.** Создайте класс *НевегетарианскаяПицца* как подкласс класса *Пицца* и объягите классы *ВегетарианскаяПицца* и *НевегетарианскаяПицца* непересекающимися (*Disjoint classes*).

1. Выберите *Пицца* в иерархии классов на вкладке «*Классы*». Нажмите кнопку «*Добавить*» значок (+), чтобы создать новый класс – подкласс *Пиццы*.

2. Назовите новый класс *НевегетарианскаяПицца*.

3. Сделайте классы *ВегетарианскаяПицца* и *НевегетарианскаяПицца* непересекающимися.

Теперь мы хотим определить класс *НевегетарианскаяПицца* как класс пицц, которые не являются элементами класса *ВегетарианскаяПицца*.

4. Убедитесь, что класс *НевегетарианскаяПицца* выбран в иерархии классов на вкладке «*Классы*».

5. В секции «*Суперкласс*» с помощью редактора выражений введите утверждение *«Пицца and (not (ВегетарианскаяПицца))»*.

6. Нажмите кнопку *OK*, чтобы создать выражение.

Очень полезная возможность редактора выражений – это возможность «автозаполнения» имен классов, имен свойств и имен индивидуальностей. Автозаполнение для встроенного Редактора выражений включается с помощью клавиши табуляции. В приведенном выше примере, если бы мы набрали в редакторе *«Be»* и нажали клавишу табуляции, то для завершения слова *Be* можно было бы сделать выбор из списка, как показано на рис. 1.73. Клавиши со стрелками вверх и вниз могут быть использованы для выбора *ВегетарианецПицца*, после чего следует нажать клавишу ввода и завершить слово. Описание класса будет иметь вид, показанный на рис. 1.74.

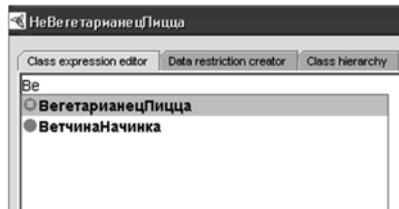


Рис. 1.73. Пример использования автозаполнения

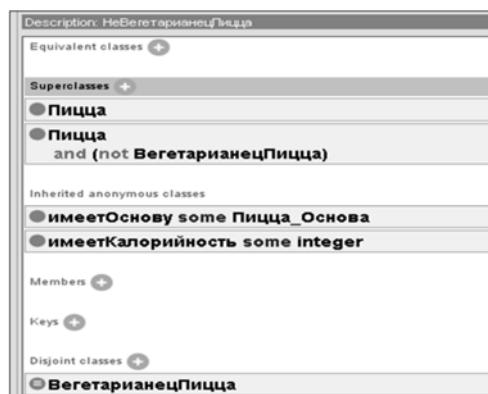


Рис. 1.74. Описание класса *НеВегетарианецПицца* как примитивного класса

Преобразуем необходимые условия в необходимые и достаточные условия, чтобы сказать: если пицца не принадлежит классу

*ВегетарианецПицца*, то она является элементом класса *НевегетарианецПицца*.

**Задание 7.** Добавьте к классу *НевегетарианецПицца* необходимые и достаточные условия.

1. Убедитесь, что выбран класс *НевегетарианецПицца* в иерархии классов.

2. Выберите пункт «Edit → Преобразовать в определяемый класс» или скопируйте и вставьте утверждение из секции «Суперкласс» в секцию «Эквивалентный класс». Описание класса *НевегетарианецПицца* будет иметь вид, изображенный на рис. 1.75.

**Задание 8.** Используйте резонер для классификации онтологии.

1. Нажмите кнопку «Синхронизировать резонер ...» на панели инструментов резонер (*Reasoner*). Как только резонер закончит свою работу, на вкладке выводимой иерархии можно просмотреть результаты классификации. Иерархия классов будет иметь вид, показанный на рис. 1.76.

**Рис. 1.75.** Описание класса *НевегетарианецПицца* как определяемого класса

Как видно, *МаргаритаПицца* и *СохоПицца* были классифицированы как подклассы *ВегетарианецПицца*. АмериканскаяПицца и АмериканскаяОстрагаяПицца были классифицированы как *НеВегетарианецПицца*. Кажется, все работает.

Однако давайте добавим пиццу, у которой нет аксиомы закрытия для свойства *имеетНачинку*.

**Задание 9.** Создайте подкласс класса *ИменованнаяПицца* с начинкой из *Моцареллы*.

1. Создайте подкласс класса *ИменованнаяПицца* и назовите его *НезакрытаяПицца*.

2. Убедитесь, что *НезакрытаяПицца* выделена в иерархии классов и выберите секцию «*Суперкласс*».

3. Нажмите кнопку «*Добавить класс*» для отображения диалогового окна редактора выражений.

4. Наберите выражение «*имеетНачинку some МоцареллаНачинка*».

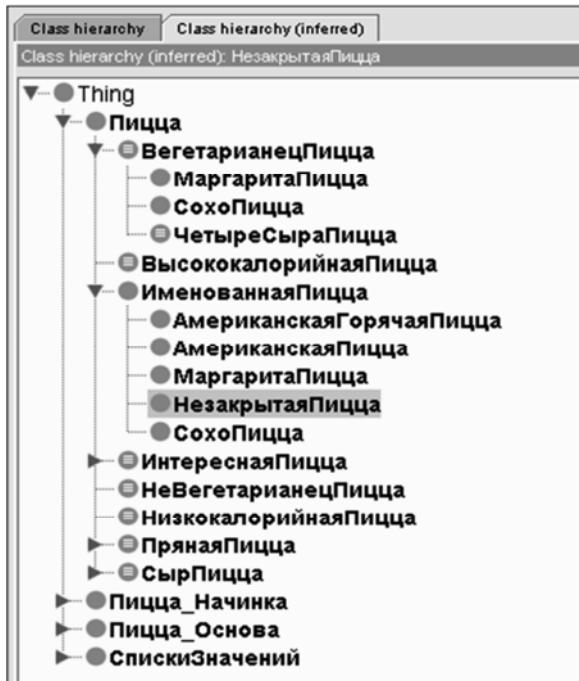


Рис. 1.76. Выводимая иерархия классов

5. Нажмите «Enter», чтобы закрыть диалоговое окно и создать ограничение.

Если пицца является членом класса *НезакрытаяПицца*, то она удовлетворяет следующим условиям: является элементом *ИменованнойПиццы* и имеет хотя бы одну начинку, которая относится к членам класса *МоцареллаНачинка*. Помните, что из-за открытого мира и потому, что не добавлена аксиома закрытия для свойства *имеетНачинку*, *НезакрытаяПицца* может иметь дополнительные начинки, не принадлежащие к виду *МоцареллаНачинка*.

Используйте резонер для классификации онтологии. Нажмите «Синхронизировать резонер...» в раскрывающемся меню резонера.

Изучите выводимую иерархию классов. Обратите внимание, что *НезакрытаяПицца* не отнесена ни к классу *ВегетарианецПицца*, ни к классу *НеВегетарианецПицца*. Как и ожидалось (из-за рассуждений в открытом мире), *НезакрытаяПицца* не была классифицирована как *ВегетарианецПицца*. Резонер не может определить, является ли *НезакрытаяПицца* – *ВегетарианецПицца*, потому что нет аксиомы о закрытии свойства *имеетНачинку* и пицца может иметь другие начинки. Можно было бы ожидать, что *НезакрытаяПицца* будет классифицирована как *НеВегетарианецПицца*, так как она не относится к классу *ВегетарианецПицца*. Тем не менее, рассуждения в открытом мире предполагают, что, так как нельзя установить, является или не является *НезакрытаяПицца* – *ВегетарианецПицца*, то поэтому нельзя ее классифицировать и как *НеВегетарианецПицца*.

### **Создание других конструкций OWL в Prot'eg'e 4**

В этой главе рассказывается, как создать некоторые другие конструкции *OWL* с использованием *Prot'eg'e 4*. Эти конструкции могут быть созданы в новых *Prot'eg'e 4* проектах, если это необходимо.

#### **Создание индивидуальностей**

*OWL* позволяет определить индивидуальности и их свойства. Индивидуальности (объекты) могут быть также использованы в описании классов, а именно, в ограничениях *HasValue* (*имеетВеличину*) и перечисляемых классах, которые будут рассмотрены в

следующем разделе. Для создания индивидуальностей в Prot'eg'e 4 используется вкладка *Индивидуальности* (*Individuals*).

Предположим, нужно описать страну происхождения для различных начинок пиццы. В первую очередь, необходимо добавить различные страны в нашу онтологию. Страны «Англия», «Италия», «Америка», как правило, рассматриваются как индивидуальности. Для этого в нашей онтологии пиццы создадим класс «Страны» и затем заполним его экземплярами стран.

**Задание 10.** Создайте класс с названием «Страны» и заполните его названиями стран.

1. Создайте класс «Страны» как подкласс *Thing*.
2. Переключитесь на вкладку *Индивидуальности* (*Individuals*), показанную на рис. 1.77, 1.78.
3. Нажмите кнопку «Добавить индивида», как показано на рис. 1.78. (Помните, что «Индивидуальность» – это «Сущность (Объект)» в терминологии онтологии ).

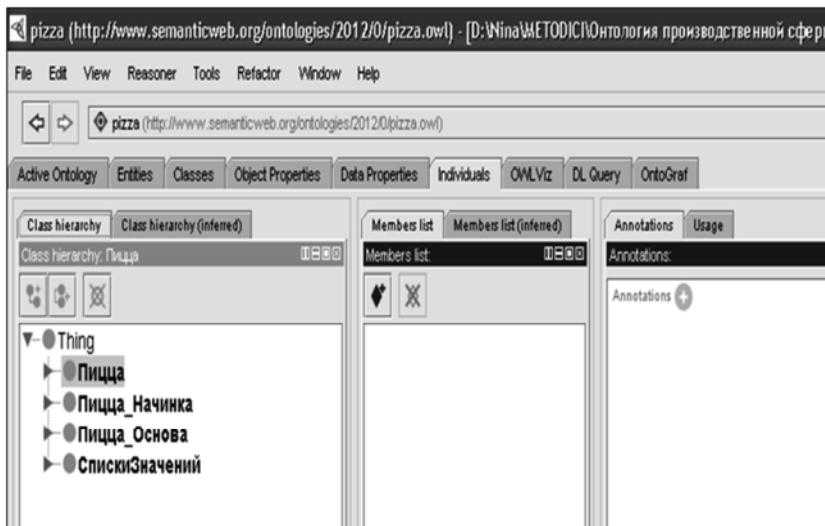


Рис. 1.77. Вкладка *Индивидуальности*

4. Задайте имя индивидуальности «Италия».
5. Выберите «Добавить» значок (+) рядом с секцией *Типы* на панели «Описание индивидуальностей», расположенной в центре

на отдельной вкладке. Выберите класс «Страны» из иерархии классов, это сделает *Италию* элементом класса «Страны».

6. Повторите действия п. 5 и создайте в классе «Страны» экземпляры *Америка*, *Англия*, *Франция* и *Германия*.

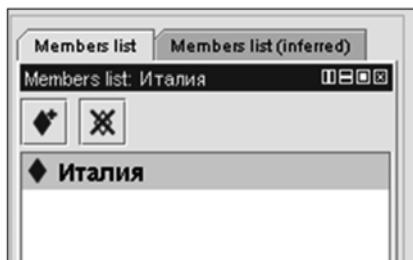


Рис. 1.78. Кнопки манипулирования индивидуальностями

### **Ограничения HasValue (имеетВеличину)**

Напомним, что в *OWL* нет понятия уникальности имени. Индивидуальности могут быть «Подобны...» или «Отличны...» от других индивидуальностей. В *Protégé 4* это можно сделать с помощью «*SameAs*» и «*DifferentFrom*» секций в «Описании индивидуальности»

Создав некоторые индивидуальности, теперь мы можем использовать их в определении классов, как описано в § 1.3–1.5.

Ограничение *HasValue* (*имеетВеличину*) описывает множество индивидов, которые имеют хотя бы одну связь по заданному свойству с конкретной индивидуальностью, указанной в ограничении. Например, ограничение *имеетСтрануПроисхождения value Италия* (где *Италия* – это индивидуальность) описывает множество индивидов (анонимный класс объектов), которые имеют хотя бы одну связь по свойству *имеетСтрануПроисхождения* со страной *Италия*.

Предположим, нужно указать происхождение ингредиентов в нашей онтологии пиццы. Например, можно было бы сказать, что сыр *Моцарелла* (*МоцареллаНачинка*) из Италии. У нас уже есть некоторые страны в нашей онтологии (в том числе Италия), которые представлены как индивиды. Можно использовать *HasValue* ограничение для *МоцареллаНачинка*, чтобы указать в качестве страны происхождения начинки *Италию*.

**Задание 11.** Создайте *HasValue* ограничение, чтобы определить Италию как страну происхождения *МоцареллаНачинка*.

1. Переключитесь на вкладку «*Object Properties*» (Свойства объектов). Создайте новое свойство объекта и назовите его *имеетСтрануПроисхождения*.

2. Переключитесь на вкладку «Классы» и выберите класс *МоцареллаНачинка*.

3. Выберите «Добавить» значок (+) в разделе «Суперкласс», чтобы открыть редактор выражений.

4. Наберите выражение «*имеетСтрануПроисхождения value Италия*» для ограничения.

5. Нажмите «Enter», чтобы закрыть диалоговое окно и создать ограничение.

Панель «Описание класса» будет иметь вид, показанный на рис. 1.79.

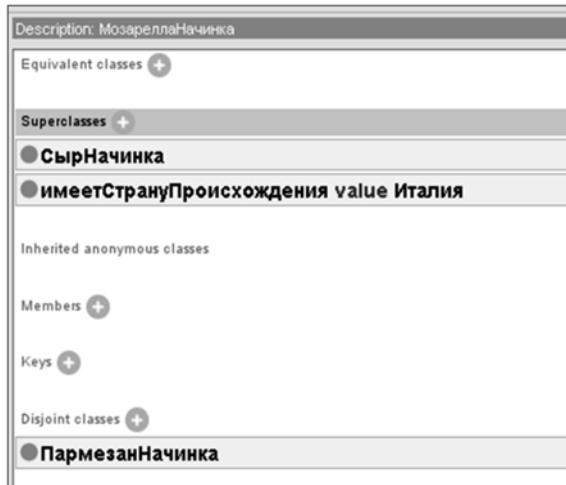
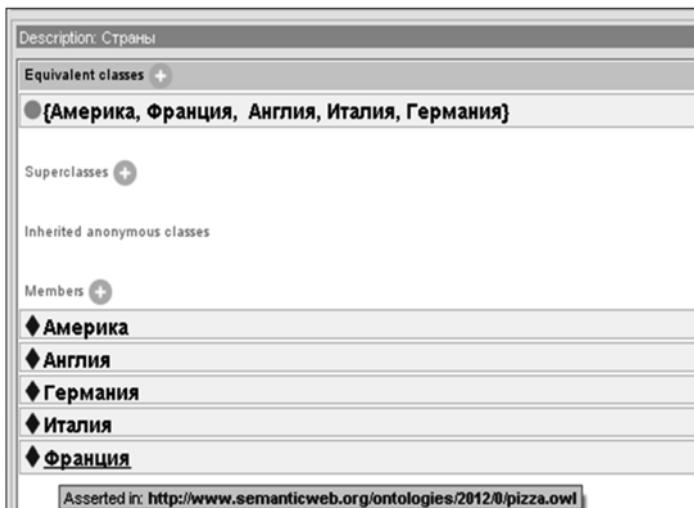


Рис. 1.79. Панель описания класса *МоцареллаНачинка*

Условия, которые были определены для *МоцареллаНачинка*, теперь говорят, что индивиды, являющиеся членами класса *МоцареллаНачинка*, также являются членами класса *СырНачинка* и связаны со страной *Италия* через свойство *имеетСтрануПроисхождения* и связаны, по крайней мере, с одним членом класса *Мягкая* через свойство *имеетПряность*. В более естественной форме это означает, что *МоцареллаНачинка* имеет сыры и появилась в Италии и является слегка пряной.

### Перечисляемые классы

До сих пор именованные классы и суперклассы определялись с помощью утверждений (ограничений). *OWL* позволяет описать классы путем задания списка индивидуальностей. Например, мы можем определить класс *ДниНедели*, который будет содержать следующие значения: *Воскресенье, Понедельник, Вторник, Среда, Четверг, Пятница и Суббота*. Такие классы называются перечисляемыми. В *Prot'eg'e 4* перечисляемые классы определяются в редакторе выражений в виде списка в фигурных скобках, где элементы разделяются запятой. Например, *{Понедельник, Вторник, Среда, Четверг, Пятница, Суббота}*. Индивидуальности должны быть до этого момента созданы в онтологии. Перечисляемые классы являются анонимными классами, к которым принадлежат только индивидуальности, указанные в перечислении. Мы можем присоединить этот список к именованному классу в *Prot'eg'e 4* путем создания перечисления в качестве Эквивалентного класса.



**Рис. 1.80.** Преобразование класса «Страны» в перечисляемый класс

**Задание 12.** Выполните преобразование класса «Страны» в перечисляемый класс.

1. Переключитесь на вкладку «Классы» и выберите класс «Страны».

2. Выберите секцию «*Эквивалентные классы*».
3. Нажмите «*Добавить*» значок (+) в секции «*Эквивалентные классы*». Появится редактор выражений, в котором наберите {*Америка, Англия, Франция, Германия, Италия*} в текстовом поле. Нажмите клавишу ввода, чтобы принять перечисление и закрыть редактор выражений. Панель «*Описание класса*» будет иметь вид, показанный на рис. 1.80.

### **Аннотация**

*OWL* позволяет аннотировать (пояснять) классы, свойства, индивидуальности и саму онтологию с помощью фрагментов информации (метаданных). Эти фрагменты могут принимать форму аудита или редакции. Например: комментарии, дата создания, автор или ссылки на ресурсы, такие как веб-страницы и т.д. *Full (Полный)* *OWL* не ставит каких-либо ограничений по использованию аннотации. Тем не менее, *OWL-DL*(*OWL* – язык описания онтологии, основанный на дескрипционной логике *DL*) накладывает ограничения на использование аннотации. Вот два наиболее важных ограничения:

1) в аннотации должны находиться либо литералы данных (например, «Михаил», 25, 3.11, 94), либо ссылка *URI* или индивидуальности;

2) свойства аннотации не могут быть использованы в аксиомах свойств – например, они не могут быть использованы в иерархии свойств, поэтому они не могут быть вложенными друг в друга. Также не должно быть домена и диапазона, установленного для них.

*OWL* имеет пять предопределенных характеристик (атрибутов, реквизитов) аннотации, которые могут быть использованы для аннотирования классов (в том числе анонимных классов, таких как ограничения), свойств и индивидуальностей:

1) **owl:versionInfo** (**информация о версии**) – диапазон значений этого свойства – строки;

2) **rdfs:label** (*RDFS*: метка) – имеет диапазон значений – строки. Это свойство может быть использовано для придания смысла именам элементов онтологии, таких как классы, свойства и индивидуальности. *RDFS*: метка может обеспечить многоязычные имена для элементов онтологии;

3) **rdfs:comment** (комментарий) – имеет диапазон значений – строки и предназначен для пояснения смысла термина;

4) **rdfs:seeAlso** – имеет диапазон значений – *URI*, которые могут быть использованы для идентификации связанных с ними ресурсов;

5) **dfs:isDefinedBy** – возможные значения – это *URI* ссылки, которые можно использовать для ссылки на онтологию, определяющие онтологию элементов, таких как классы, свойства и индивидуальности.

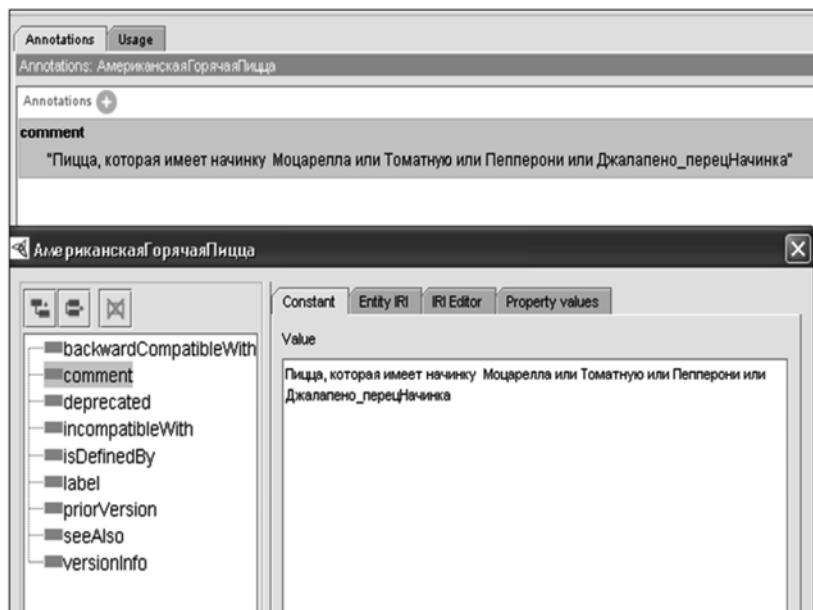


Рис. 1.81. Панель аннотации

Например, аннотация **rdfs:comment**: – это комментарий, поясняющий назначение классов в *Prot'eg'e 4*. Аннотация **rdfs:label** может быть использована для обеспечения альтернативных имен для классов, свойств и т.д. Есть также несколько свойств (атрибутов аннотации), которые можно использовать для аннотирования онтологии. Все эти атрибуты перечислены ниже и представляют собой ссылки *URI*, с помощью которых можно обратиться к другой онтологии. Можно также использовать реквизит **owl:versionInfo**

для аннотирования онтологии, с помощью которого описывается версия текущей онтологии:

– **owl:priorVersion** – идентифицирует предыдущие версии онтологии;

– **owl:backwardsCompatibleWith** – определяет предыдущую версию онтологии, совместимую с текущей онтологией. Это означает, что все идентификаторы из предыдущей версии имеют одинаковое предполагаемое значение в текущей версии. Следовательно, любая онтология или приложение, которое ссылается на предыдущую версию, может смело переключаться на ссылку к новой версии;

– **owl:incompatibleWith** – определяет предыдущую версию онтологии, не совместимую с нынешней онтологией.

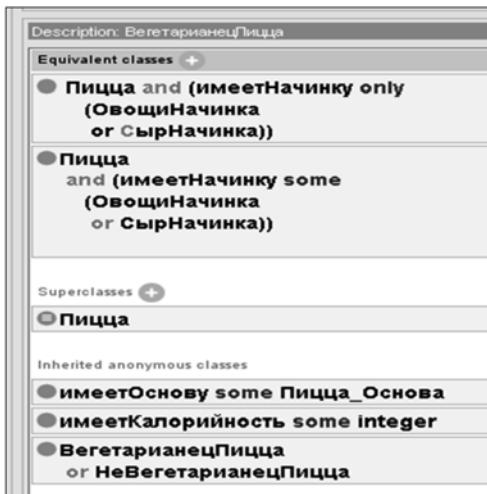
Для создания аннотации используется панель «Аннотация», расположенная на каждой из вкладок «Активная Онтология», «Классы», «Свойства объектов» и «Свойства данных». Можно управлять аннотацией, используя вкладки окна редактирования аннотации. Новая аннотация может быть создана нажатием кнопки *Annotations* (+) на панели «Аннотация». Процесс создания аннотации иллюстрируется рис. 1.81.

### Несколько наборов необходимых и достаточных условий

В OWL можно иметь несколько наборов необходимых и достаточных условий, как показано на рис. 1.82.

В поле «Описание класса» несколько наборов необходимых и достаточных условий представлены с использованием заголовка «Эквивалентные классы» для каждого набора необходимых и достаточных условий. Чтобы создать новый набор необходимых и достаточных условий, используйте «Добавить» значок (+) рядом с заголовком «Эквивалентные классы». Эквивалентные классы могут быть описаны в появляющемся при этом диалоговом окне «Описание класса».

Когда несколько ограничений используется, общее описание получается путем выполнения операции пересечения отдельных ограничений.



**Рис. 1.82.** Необходимые условия, а также несколько наборов Необходимых и достаточных условий

### Контрольные вопросы

1. Что такое *Data Properties*? Какие связи описывают эти свойства?
2. Как создать связи с типизированными характеристиками?
3. Как описать классы *НизкокалорийнаяПицца* и *ВысококалорийнаяПицца*?
4. Как задать ограничения на значения типизированных данных (характеристик *Data Properties*)?
5. Как вы понимаете термин «Рассуждения в открытом (закрытом) мире»?
6. Что такое индивидуальность? Как ее создать? Как задать характеристики и свойства индивидуальности?
7. Каково назначение ограничения «HasValue»? Приведите пример такого ограничения.
8. Как создать перечисляемые классы?
9. Для чего нужна аннотация? Какие фрагменты информации она содержит?
10. Как трактуются несколько наборов необходимых и необходимых и достаточных условий?

## Подведем итоги

Подведем итог описанному процессу создания онтологии пиццы.

1. Любой класс описывается с помощью ограничений на свойства элементов класса.
2. Под свойствами понимаются:
  - характеристики связей объектов класса с другими объектами своего или другого класса;
  - связь с атрибутами объекта (задаются типизированными данными).
3. Ограничения задаются с помощью логических формул.

## Типы ограничений

Все виды ограничений описывают множество, которое может содержать индивидуальности. Это множество может рассматриваться как *анонимный* класс. Любые объекты, которые являются членами этого анонимного класса, удовлетворяют ограничениям, описывающим класс. Утверждения описывают ограничения на отношения, в которых объекты могут участвовать.

*Когда описывается именованный класс с помощью ограничений, то этот класс становится подклассом анонимного класса, соответствующего введенным ограничениям.*

## Ограничение с помощью квантора

Ограничение с помощью квантора состоит из трех частей.

1. Квантор – либо существования  $\exists$  [**some** (некоторый)], либо всеобщности  $\forall$  [**only** (только)].

2. Свойство (отношение), по которому создается ограничение.

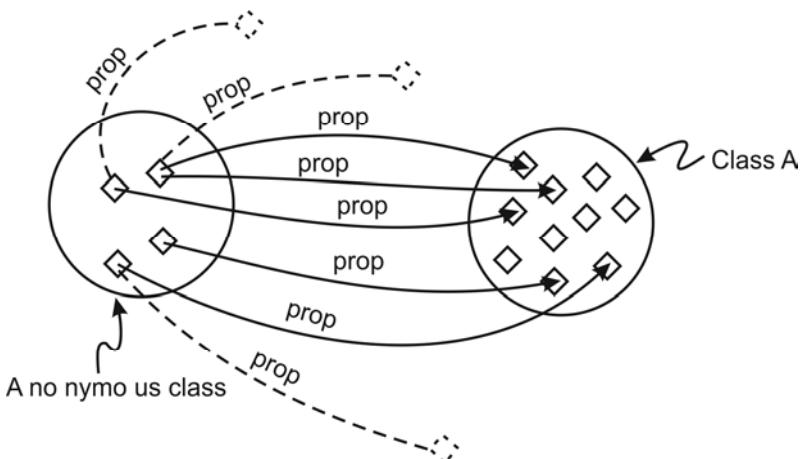
3. Наполнитель (Правый аргумент отношения), который описывает класс (либо именем, либо выражением).

Для элемента класса квантор эффективно накладывает ограничения на отношения, в которых этот элемент участвует. Он делает это, либо, указав что, по крайней мере, один экземпляр заданного отношения должен существовать, либо путем указания, что существовать могут только экземпляры заданного вида отношений.

### Экзистенциальные ограничения *someValuesFrom*

Экзистенциальные ограничения, также известные как «*some ValuesFrom*» ограничения, или «*some*» ограничения, обозначаются в *DL*-синтаксисе знаком Э. Экзистенциальные ограничения описывают множество сущностей, которые имеют, по крайней мере, один экземпляр заданного отношения с объектами, являющимися членами определенного класса.

Рис. 1.83 показывает смысловую схему экзистенциального ограничения *prop some ClassA* – то есть ограничение по свойству *prop* с наполнителем *ClassA*.



**Рис. 1.83.** Смысловая схема экзистенциальных ограничений  
*(prop some ClassA)*

Обратите внимание, что все сущности анонимного класса, описанного ограничением, связаны отношением *prop*, по крайней мере, с одним элементом, являющимся членом класса *ClassA*. Пунктирные линии на рис. 1.83 показывают тот факт, что сущность может иметь отношение *prop* с другими сущностями, не являющимися членами класса *ClassA*, хотя это не было оговорено. Экзистенциальное ограничение не ограничивает отношение только членами класса *ClassA*, оно просто гласит, что каждый объект должен иметь хотя бы один экземпляр отношения с членом *ClassA* – это предположение об открытости мира (OWA).

Для более конкретного примера: экзистенциальное ограничение «имеетНачинку some МоцареллаНачинка» описывает множество индивидов, которые участвуют как минимум в одном *имеетНачинку* отношении с другим индивидом, являющимся членом класса *МоцареллаНачинка*, – в более естественной форме это звучит как описание пиццы, имеющей начинку типа *МоцареллаНачинка*.

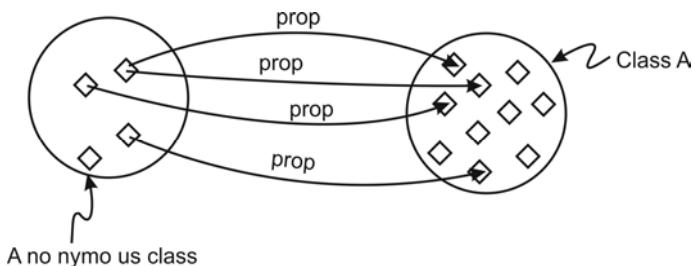
Тот факт, что используется экзистенциальное ограничение для описания группы индивидов, которые имеют хотя бы одно отношение *имеетНачинку* с членом класса *МоцареллаНачинка*, не означает, что эти сущности не могут быть связаны с другими видами начинок.

### Универсальные ограничения *allValuesFrom*

Универсальные ограничения, также известные как «*allValuesFrom*» ограничения, или «*only*» ограничения, определяют наполнитель в виде указанного в ограничении класса. Универсальные ограничения обозначаются символом  $\forall$ . Универсальные ограничения описывают набор индивидов, которые имеют данный вид связи только с элементами указанного в ограничении класса.

Особенностью универсальных ограничений является то, что они сообщают, что объекты, удовлетворяющие этому ограничению, не могут участвовать в связи ни с какими объектами, не являющимися членами указанного класса.

Универсальное ограничение по свойству *prop* с наполнителем из элементов класса *ClassA* изображено на рис. 1.84. Важно отметить, что универсальные ограничения не «гарантируют» существования экземпляров заданного отношения. Они просто утверждают, что если такой вид отношения существует, то экземпляры его должны связывать элементы анонимного класса только с членами класса *ClassA*.



**Рис. 1.84.** Схематическое изображение универсального ограничения *prop*

Давайте посмотрим на пример универсальных ограничений. Ограничение «имеетНачинку **only** ПомидорыНачинка» описывает анонимный класс объектов, у которых есть начинки, принадлежащие только классу *ПомидорыНачинка*, или, объектов, не участвующих ни в каких *имеетНачинку* отношениях вовсе.

### Объединение кванторов существования и всеобщности

Наиболее общий шаблон описания ограничений представляет собой комбинация экзистенциальных и универсальных ограничений.

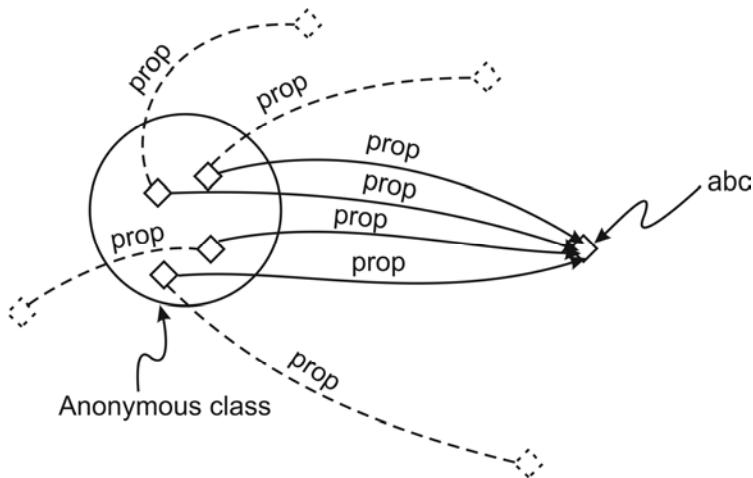
Например, следующие два ограничения могут быть использованы вместе: «имеетНачинку **some** МоцареллаНачинка», а также «имеетНачинку **only** МоцареллаНачинка». Такая комбинация описывает множество объектов (пицц), у которых есть хотя бы одна связь *имеетНачинку* с *МоцареллаНачинка*, и в то же время начинка у этих объектов (пицц) может быть только из класса *МоцареллаНачинка*. В отличие от схемы, показанной на рис. 1.83, в анонимном классе *все* элементы будут связаны с элементами только класса *МоцареллаПицца*.

Стоит отметить, что ошибочно при описании класса использовать только универсальные ограничения по данному свойству без использования «соответствующих» экзистенциальных ограничений. В приведенном выше примере, если бы использовались только универсальное ограничение «имеетНачинку **only** МoцареллаНачинка», то было бы описано множество пицц, которые связаны отношением *имеетНачинку* только с членами класса *МоцареллаПицца*, а также тех элементов (пицц), у которых вообще нет начинки, т.е. они не связаны отношением *имеетНачинку* ни с одним элементом.

### Ограничения HasValue(ИмеетЗначение)

Ограничение *HasValue (value)* описывает анонимный класс объектов, связанных некоторым отношением с конкретной (заданной своим именем) индивидуальностью определенного класса. Сравним квантифицированные ограничения и ограничения *HasValue*. Первые задают связь объектов с некоторыми (какими-то) элементами заданного класса, вторые ограничения связывают объекты с одним заданным элементом указанного класса.

Рис. 1.85 показывает схематическое изображение *HasValue* ограничения ргор с индивидуальностью *abc*. Это ограничение описывает анонимный класс объектов, которые имеют, по крайней мере, одну связь по свойству *prop* с конкретной индивидуальностью *abc*. Пунктирные линии на рис. 1.85 показывают, что могут существовать связи и с другими объектами, но должна существовать хотя бы одна связь с *abc*. Следует отметить, что *HasValue* ограничения семантически эквивалентны экзистенциальным ограничениям по тому же свойству.



**Рис. 1.85.** Схематическое изображение ограничения  
*HasValue*: «*prop value abc*»

### Мощность ограничения

Мощность ограничения используется для описания количества экземпляров заданного отношения, в которых объект может присутствовать.

Мощность ограничения концептуально легче понять, чем ограничения-кванторы. Существует три варианта мощности ограничений: *Минимальная мощность ограничения*, *Максимальная мощность ограничения* и просто *Мощность ограничения*.

### Минимальная мощность ограничения

*Минимальная мощность ограничения (min)* указывает минимальное количество экземпляров отношений, в которых объект

должен принимать участие для данного свойства. Символом для минимальной мощности ограничения является «больше или равно» ( $\geq$ ) (*min*). Например, ограничение мощности «имеетНачинку *min* 3» описывает объекты (анонимный класс, содержащий объекты), каждый из которых имеет, по меньшей мере, три экземпляра отношения *имеетНачинку*. Минимальная мощность ограничения не накладывает никаких ограничений на максимальное число экземпляров данного отношения. Всего экземпляров может быть  $\geq$  минимальному значению.

### **Максимальная мощность ограничения**

*Максимальная мощность ограничения* указывает максимальное количество экземпляров отношений, в которых объект должен принимать участие для данного свойства. Символ для ограничения максимальной мощности является «менее или равно» ( $\leq$ ). Например, ограничение мощности «имеетНачинку *max* 2» описывает объекты (анонимный класс, содержащий объекты), которые имеют не более чем два экземпляра отношения *имеетНачинку*. Обратите внимание, что максимальная мощность ограничения не накладывает никаких ограничений на минимальное число экземпляров данного отношения. Всего экземпляров может быть  $\leq$  максимальному значению.

### **Точное значение мощности ограничения**

Точное значение мощности ограничения указывает конкретное количество экземпляров отношений, в которых объект должен участвовать для данного свойства. Символ точного значения мощности «равно» (=) (*exactly*). Например, точное значение мощности ограничения «имеетНачинку *exactly* 5» описывает множество объектов (анонимный класс индивидов), которые участвуют ровно в пяти экземплярах отношения *имеетНачинку*.

### **Присвоение уникального имени и ограничений мощности**

*OWL* не использует понятия уникальности имени (УНА). Это означает, что различные имена могут относиться к одним и тем же лицам, например имена «*Миша*» и «*Михаил*» могут относиться к одной и той же особи (или не могут). Мощность ограничения полагается на «подсчет» индивидов, поэтому важно указать, что

либо «Миша» и «Михаил» одно и то же лицо, либо, что они отличны друг от друга. Предположим, что индивид «Николай» связан с индивидами «Миша», «Михаил» и «Михаил Корякин» через свойство *работаетВместеС*. Было также заявлено, что индивид «Николай» является членом категории лиц, которые работают с максимум двумя другими лицами. Так как OWL не использует уникальности имен, то здесь не возникнет ошибки, скорее будет сделан вывод, что двое из трех лиц являются одним и тем же лицом. Некоторые резонеры (например, *RACER*) используют присвоение уникальных имен! Если «Миша», «Михаил» и «Михаил Корякин» будут объявлены как различные лица, то это сделает Базу знаний противоречивой.

### Комплексное описание класса

Класс *OWL* описывается в терминах своего суперкласса. Эти суперклассы, как правило, образуются на основе именованных классов и ограничений, которые на самом деле задают анонимные классы. Описание суперкласса может принимать форму «комплексного (сложного) описания». Эти сложные описания могут быть созданы с помощью более простых описаний классов, которые соединены логическими операторами. В частности:

- *И (and)* – класс, полученный с помощью оператора **and**, является пересечением отдельных классов.
- *ИЛИ (or)* – класс, полученный с помощью оператора **or**, является объединением отдельных классов.

#### Пересечение классов (*and*)

Пересечение классов описывается путем комбинации двух или более классов, используя оператор *И (and)*. Рассмотрим, например, пересечение классов *Человек* и *МужскойРод*, результат изображен на рис. 1.86. Это анонимный класс, который содержит людей, являющихся одновременно членами класса *Человек* и класса *МужскойРод*. Семантика операции пересечения классов означает, что результирующий анонимный класс является подклассом класса *Человек* и подклассом класса *МужскойРод*.

Объявленный как пересечение анонимный класс может быть использован для описания другого класса. Например, предположим, нужно построить описание класса *Мужчина*. Можно указать, что *Мужчина* является подклассом *Анонимного класса*, описывае-

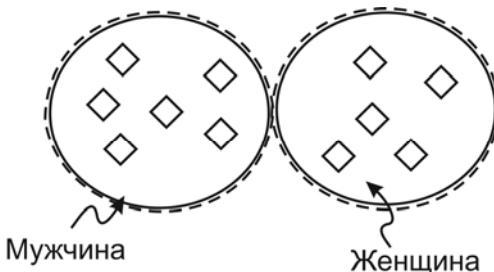
мого как пересечение классов *Человек* и *МужскойРод*. Другими словами, *Мужчина* является подклассом класса *Человек* и класса *МужскойРод*.

### Объединение классов (or)

Объединение классов создается путем объединения двух или более классов с использованием оператора *ИЛИ* (*or*). Например, рассмотрим объединение классов *Мужчина* и *Женщина*, результат изображен на рис. 1.86. Он описывает анонимный класс, который содержит людей, принадлежащих либо классу *Мужчина*, либо классу *Женщина* (либо обоим).



**Рис. 1.86.** Пересечение классов *Человек and (И) МужскойРод* – заштрихованная область представляет пересечение



**Рис. 1.87.** Объединение классов *Мужчина or (ИЛИ) Женщина*

Описанный как объединение анонимный класс может быть использован в определении другого класса. Например, класс *Личность* может быть эквивалентом объединения классов *Мужчина* и *Женщина*.

**Контрольные вопросы**

1. Что такое анонимный класс?
2. Каков смысл различных типов ограничений?
3. Когда нужно объединять кванторы существования и всеобщности?
4. Что такое минимальное и максимальное ограничение мощности?
5. Каков смысл ограничения HasValue?

**Контрольные задания**

1. Добавьте в онтологию пиццы ее классификацию по следующим признакам:
  - по размеру;
  - по стоимости;
  - по стране происхождения;
  - по виду начинок;
  - по популярности.
2. Разработайте онтологии для рекламы следующих товаров:
  - шоколада;
  - аквариумных рыбок;
  - канцелярских принадлежностей;
  - мобильных телефонов;
  - гаджетов;
  - мебели для кухни;
  - косметики.

*Просто невероятно, как сильно могут повредить правила, едва только наведешь во всем слишком строгий порядок.*

*Георг Кристофер Лихтенберг*

## Глава 2

### ТЕХНОЛОГИЯ РАЗРАБОТКИ ОНТОЛОГИИ ПРЕДМЕТНОЙ ОБЛАСТИ

В предыдущем разделе в процессе разработки фрагмента онтологии Пиццы вы познакомились с основными элементами онтологии. Это классы, подклассы, объектные свойства, свойства данных, индивиды и ограничения в виде аксиом.

Процесс создания онтологии предметной области можно формализовать как последовательность шагов, т.е. тем самым предложить один из вариантов технологий ее разработки (рис. 2.1).

Концептуализация



**Рис. 2.1.** Жизненный цикл создания онтологии предметной области

В настоящее время существуют разные подходы к проектированию онтологий [8, 21, 24, 37]. Это говорит о том, что сама технология разработки зависит от предметной области. В этом пособии рассмотрены две методологии: методологию построения онтологии

продукта (пример – онтология пиццы) и методологию разработки онтологии на основе концептуальной модели предметной области. Материал данного раздела сформирован на основе следующих источников [21, 24, 25].

## 2.1. Методология построения онтологии продукта

Задача создания онтологии *Пиццы* может быть сформулирована в обобщенном виде следующим образом. Имеется некоторый продукт (сущность), для которого необходимо создать описание его свойств и характеристик, отразив в этом описании наиболее важные и существенные черты.

Сначала определяем атомарные (базовые) элементы онтологии: примитивные классы и роли (объектные свойства и свойства данных).

На первом этапе выделяем *составные части продукта (отношение часть-целое)* или основные характеристики, отражающие его ценность для потребителя информации. Например (рис. 2.2), *Пицца* имеет две составные части: основу и начинку, от состава и приготовления которых зависят ее вкусовые качества, *Автомобиль* имеет двигатель, корпус, тормозную систему и т.д., от вида которых зависят характеристики автомобиля.

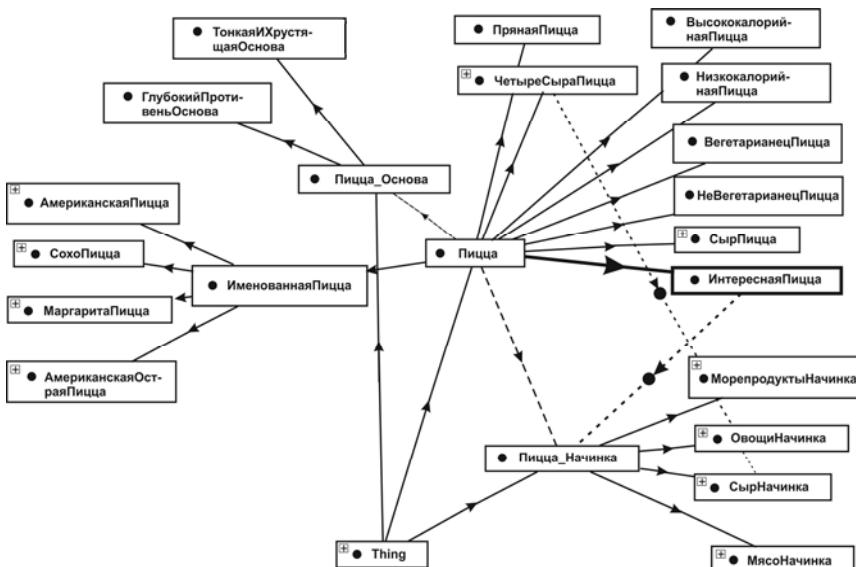


Рис. 2.2. Фрагмент онтологии *Пиццы* (Онтограф)

Различные варианты реализации каждой составной части могут быть представлены в виде иерархии классов, подклассов. Так, например, для основы пиццы два варианта ее приготовления были представлены двумя подклассами *ТонкаяИХрустящая* и *ГлубокийПротивень*, а различные варианты начинки были сгруппированы по видам продуктов *СырНачинка*, *МясоНачинка* и т.д., а затем в каждом из видов объявлены их представители *МоцареллаНачинка*, *ЛукНачинка* и т.д. в виде подклассов.

Все созданные разработчиком классы и их иерархия называются примитивными, они полностью зависят от взгляда разработчика на предметную область.

Далее определяются связи между составными частями продукта. С их помощью можно описывать его свойства. Это отношения *имеетОснову*, *имеетНачинку*, *имеетИнгредиенты*, *имеетКалорийность*.

Определив базовые элементы, можно создавать и описывать свойства классов. Если целью создания онтологии является показать информацию о продукте (например, сделать рекламу), то сначала создается список наименований продукта (номенклатура, меню) в виде подклассов. В нашем примере был создан список наиболее известных наименований пицц в виде класса *ИменованнаяПицца*.

Для каждого наименования в секции *SubClasses Of* описываются с помощью экзистенциональных или универсальных ограничений *необходимые условия* принадлежности элементов классу. *Необходимые* потому, что, создавая класс, мы знаем, какими свойствами обладают его элементы. Свойства описываются как ограничения на связи продукта с его составными частями.

Цель следующего этапа разбить на подгруппы (подклассы) (организовать по признакам) все созданные в Списке наименований разновидности продукта. Каждая подгруппа характеризуется определенными признаками. Эти признаки описываются в секции *Equivalent Classes* с помощью *необходимых и достаточных условий*, в соответствии с которыми элемент, удовлетворяющий этим условиям, должен быть отнесен к описываемому классу. В результате получаем выводимую иерархию, которая строится с помощью резонера (машины вывода). Так, например, в примере с *Пиццей* были созданы классы *ВегетарианскаяПицца* (пиццы с начинкой только овощи или

сыр), *ИнтереснаяПицца* (пицца с числом начинок больше 3), *ВысококалорийнаяПицца* (пицца с числом калорий больше 400) и т.п.

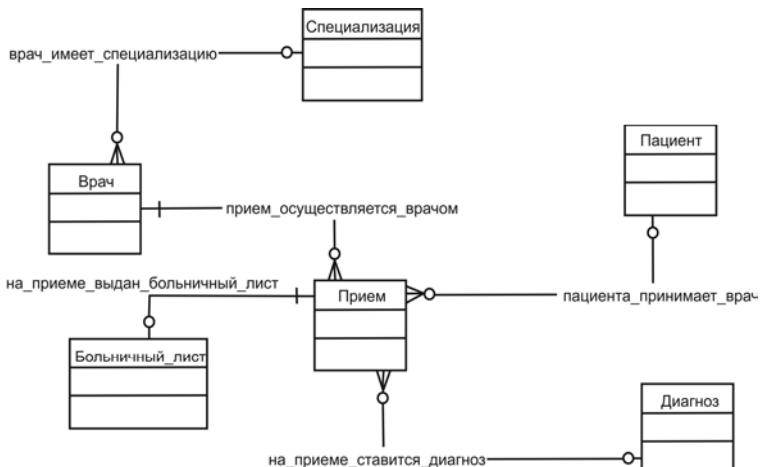
Как необходимые, так и необходимые и достаточные условия, описывающие признаки принадлежности к классу, задаются с помощью аксиом. Правила формирования аксиом рассмотрены в предыдущем разделе.

Затем можно заносить индивидуальности – примеры конкретных продуктов, описывать их свойства и выполнять классификацию с помощью резонера.

## 2.2. Проектирование онтологии на основе концептуальной модели предметной области

### Технология проектирования онтологии на основе концептуальной модели предметной области

Процесс создания онтологий [7] – сложный и трудоемкий. Существуют различные технологии разработки онтологий: снизу вверх, сверху вниз, на основе *ER* («сущность-связь») – диаграммы, на основе модели «вход-выход» [2] и т.д. В данном пособии рассматривается методика создания онтологии на основе концептуальной модели (*ER*-диаграммы) предметной области. Для примера взята предметная область «Прием врачами пациентов».



**Рис. 2.3.** *ER* – диаграмма предметной области «Прием врачами пациентов»

Проектирование ведется в редакторе онтологий *Protege 4* и включает следующие шаги.

**Шаг 1.** Сначала разрабатывается *ER* – диаграмма (диаграмма «сущность-связь») предметной области. В ней сущности предметной области отражаются поименованными прямоугольниками, связи – поименованными ребрами. Для каждой связи определяется степень связи: 1:1, 1:N, N:1, M:N. Для каждой сущности – класс принадлежности (обязательный/ необязательный) данной сущности к связи.

Например, предметную область «Прием врачами пациентов» можно описать диаграммой, полученной в среде *Sybase PowerDesigner* [4] и представленной на рис. 2.3.

**Шаг 2.** Каждой сущности ставится в соответствие класс с таким же именем (*Врач*, *Специализация*, *Пациент* и т.д.). Классы создаются в редакторе *Protege 4* на вкладке *Classes*. Имена классов рекомендуется писать с большой буквы. Если имя состоит из нескольких слов, то слова, либо разделяются знаком подчеркивания, либо пишутся слитно, при этом каждое новое слово с большой буквы, например: *Больничный\_лист*, *БольничныйЛист*. Все базовые классы являются подклассами класса *Thing* и объявляются непересекающимися, каждый из них описывает различные по свойствам сущности (рис. 2.4). Отношению «класс А является подклассом В» соответствует логическая формула  $A \rightarrow B$  (если А, то В). Это отношение может задаваться в редакторе разработчиком или с помощью ограничений (см. шаг 5). В онтологии отношение  $A \rightarrow B$  является наиболее важным.  $A \rightarrow B$  означает, что если элемент (индивидуальность) принадлежит классу *A*, то он также принадлежит классу *B*. Класс *A* может обладать многими свойствами, но среди них обязательно должны присутствовать свойства, характерные классу *B*.

**Шаг 3.** Затем на вкладке *Object Properties* определяются базовые бинарные отношения (рис. 2.5), соответствующие отношениям *ER* – диаграммы. Так как это бинарные отношения, то их имена удобно задавать в нотации <имя сущности 1\_имя\_сущности 2> или <имя сущности1\_имя\_связи\_имя\_сущности 2>, где <имя\_связи> – это название отношения, а <имя\_сущности 1> и <имя\_сущности 2> – это имена связываемых данным отношением сущностей. Например: *врач\_имеет\_специализацию*, *прием\_ведет\_врач*, *приемПациент*, *приемБоль-*

ничный\_лист и т.д. Для базовых отношений можно задать характеристики (*Functional*, *Transitive*, *Symmetric* и т.д.).

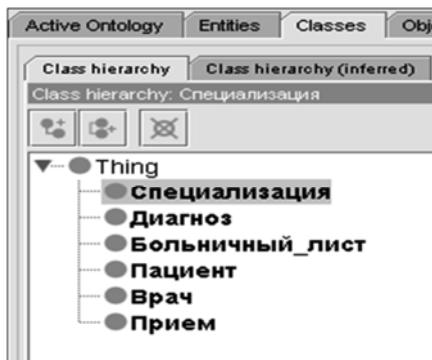


Рис. 2.4. Определение базовых классов

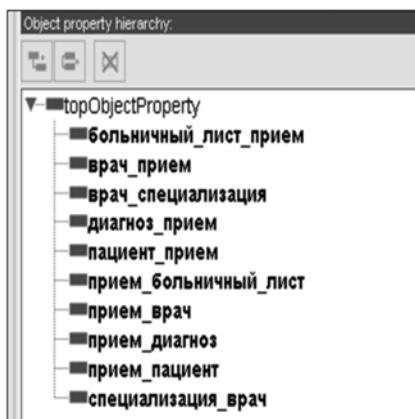


Рис. 2.5. Бинарные связи между концептами предметной области

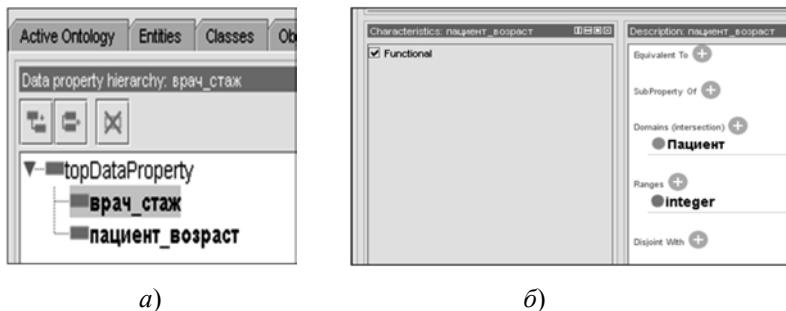
На основе базовых отношений можно получить производные от них отношения (рис. 2.5). Это инверсные отношения *врач\_прием*, *диагноз\_прием*, *больничный\_лист\_прием* и т.д. Они создаются путем указания для них свойства *Inverse Properties*. Отношения-цепочки *врач\_больничный\_лист*, *пациент\_врач*, *пациент\_диагноз* и т.д. определяются в разделе *Property chains* как композиция отношений (*o*) следующим образом:

*врач\_пациент*  $\leftarrow$  *врач\_прием o прием\_пациент*;  
*диагноз\_больничный\_лист*  $\leftarrow$

*диагноз\_прием о прием\_больничный\_лист;*  
*пациент\_больничный\_лист ←*  
*пациент\_прием о прием\_больничный\_лист.*

Производные отношения позволяют в более естественной и краткой форме описывать свойства классов с помощью логических формул.

**Шаг 4.** На вкладке *Data Properties* (рис. 2.6) описываются связи элементов классов со своими характеристиками (атрибутами), задаваемыми типизированными данными. Правило именования связей аналогично шагу 3. Например: *пациент\_возраст*, *пациент\_вес*, *врач\_стаж*, *врач\_имеет\_телефон* и т.п. Для каждой характеристики необходимо в правой части вкладки *Data Properties* указать тип ее значения и является ли она функцией элемента (*Functional*) (рис. 2.6 б).



**Рис. 2.6.** Описание атрибутов (характеристик) сущностей (а); описание типа значений характеристик сущностей (б)

**Шаг 5.** На этом шаге осуществляется описание свойств классов с помощью логических формул. В результате создается логическая модель предметной области и онтология превращается в семантическую сеть, где отражены те же сущности и связи, что и в ER – диаграмме. На рис. 2.7 показана онтология в виде онтографа (вкладка *OntoGraf*) до описания базовых классов с помощью логических формул, а на рис. 2.8 – после.

В *Protege 4* в качестве синтаксиса языка описания формул принят *Манчестерский синтаксис*. Он наиболее прост для понимания.

Рассмотрим правила описания классов на языке логики. Описание может быть сделано в форме необходимых условий (в секции *SubClasses Of*) и в форме необходимых и достаточных условий (в

секции *Equivalent classes*). В случае необходимых условий описание говорит, что если элемент принадлежит классу, то он обладает указанными свойствами. Класс, который удовлетворяет только необходимым условиям, известен как **примитивный** класс. Например, если ограничение *пациент\_прием some Прием* поместить в секцию *SubClasses Of* раздела описания класса *Пациент*, то это будет соответствовать логической формуле *Пациент → пациент\_прием some Прием* и читаться следующим образом: «Если сущность принадлежит классу *Пациент*, то она обязательно связана, хотя бы с одним приемом». То есть *Пациент* является подклассом анонимного класса, заданного ограничением, описанным в секции *SubClasses Of*.

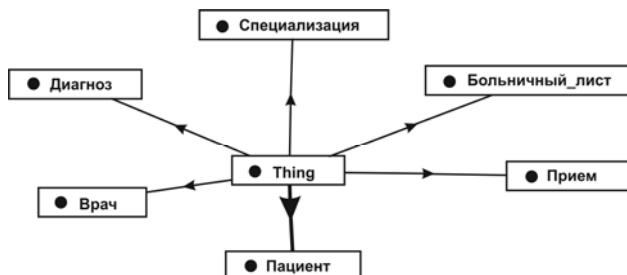


Рис. 2.7. Онтология до описания классов логическими формулами

В случае необходимых и достаточных условий описание, кроме вышеуказанного смысла, несет еще информацию, что если элемент (возможно принадлежащий другому классу) обладает заданными свойствами, то он относится также и к описываемому классу. Класс, который имеет, по крайней мере, одно утверждение о необходимых и достаточных условиях, называется **определяемым** классом. Например, если то же ограничение перенести в секцию *Equivalent classes* раздела описания класса *Пациент*, то это будет соответствовать логической формуле *Пациент ≡ пациент\_прием some Прием* и читаться следующим образом: «Если пациент, то он обязательно связан хотя бы с одним приемом, и если имеется сущность, побывавшая хотя бы на одном приеме у врача, то это пациент».

Благодаря необходимым и необходимым и достаточным условиям машина вывода (резонер) экземпляры общего класса *Thing* разделяет на группы (классы), т.е. осуществляет классификацию объектов предметной области (см. шаг 6).

Описание классов в виде логических формул несет информацию о связях элементов классов между собой, и эта связь рассматривается как свойство или характеристика экземпляра класса. В описании должны присутствовать имя связи, класс принадлежности сущности связи и степень (кардинальность) связи. Пример: *Больничный\_лист ≡ Thing and (больничный\_лист прием some Прием)*. Смысл данного описания: больничный лист – это сущность, которая обязательно связана хотя бы с одним приемом врача.

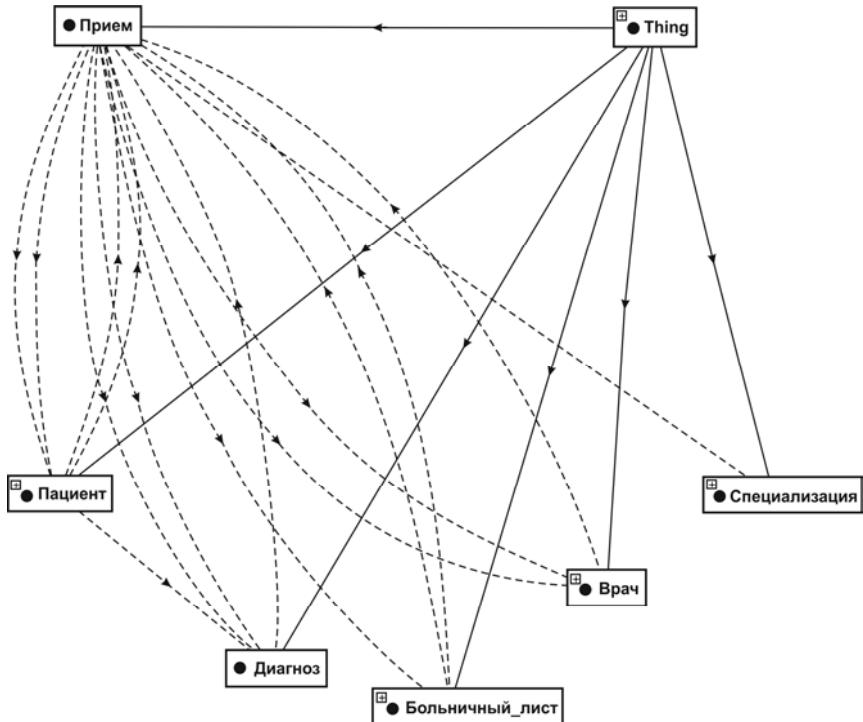


Рис. 2.8. Онтология после описания классов логическими формулами

Для описания связей сущности с элементами других классов используются ключевые слова **some** (некоторый – квантор существования), **only** (только – квантор всеобщности).

Формула *Прием → прием\_диагноз some Диагноз* имеет смысл: на приеме обязательно ставится хотя бы один диагноз. Такое описание сообщает, что любой экземпляр класса *Прием* обязательно

связан хотя бы с одним элементом класса *Диагноз*. Получаем **обязательный класс принадлежности** сущности связи. Такой способ описания (с помощью **some**) в онтологии является наиболее распространенным.

Формула *Прием → прием\_больничный\_лист only Больничный\_лист* имеет смысл: прием – это сущность, которая связана связью *прием\_больничный\_лист* **только** с элементами класса *Больничный\_лист*, при этом существуют такие экземпляры сущности *Прием*, которые вообще не связаны с классом *Больничный\_лист*. Это означает, что на приеме не обязательно выдается больничный лист. Таким образом, получаем необязательный класс принадлежности сущности связи.

Для описания кардинальности связи используются слова **min**, **max**, **exactly**. Например, формула *Прием → (прием\_больничный\_лист min 0 Больничный\_лист) and (прием\_больничный\_лист max 1 Больничный\_лист)* сообщает о том, что связь *прием\_больничный\_лист* имеет кардинальность 0..1.

Знаки  $\equiv$ ,  $\rightarrow$ , **and**, **or**, **some**, **only** – соответствуют логическим операциям: эквивалентность, импликация (если ..., то ), конъюнкция (и), дизъюнкция (или), квантор существования, квантор всеобщности.

Связь с атрибутами задается с помощью *Data Properties*. При этом можно задавать ограничения  $<=$ ,  $>=$ ,  $<$ ,  $>$ ,  $(=)$  на значения атрибутов. Например:

*Молодые ≡ пациент\_возраст some integer[< 30];*

*Пожилые ≡ пациент\_возраст some integer[>= 60];*

*Средних\_лет ≡ (not (Молодые or Пожилые)).*

На основе этих правил можно дать следующие рекомендации по их использованию.

Базовые классы и их взаимосвязь следует описывать с помощью логических формул в разделе *SubClassOf*, т.е. базовые классы нужно вначале определить как примитивные. Действительно, пока мы можем только утверждать, что если элемент относится к базовому классу, то он обладает указанными свойствами.

Все связи вначале определить с помощью ключевого слова **only**, затем для случаев обязательного класса принадлежности сущности связи добавить ограничение с ключевым словом **some**.

На рис. 2.9 приведены примеры описания базовых классов как примитивных в секции *SubClassOf*.

**a)**

Description: Больничный_лист
Equivalent To +
SubClass Of +
● Thing
● Больничный_лист_прием only Прием
● Больничный_лист_прием some Прием

**б)**

Description: Прием
Equivalent To +
SubClass Of +
● Thing
● прием_больничный_лист only Больничный_лист
● прием_врач only Врач
● прием_врач some Врач
● прием_врач_специалист only Специализация
● прием_диагноз only Диагноз
● прием_диагноз some Диагноз
● прием_пациент only Пациент
● прием_пациент some Пациент

Asserted in: <http://www.semanticweb.org/ontologies/2012/5/Прием139217002265.owl>

**в)**

Description: Пациент
Equivalent To +
SubClass Of +
● Thing
● пациент_больничный_лист only Больничный_лист
● пациент_врач only Врач
● пациент_диагноз only Диагноз
● пациент_прием only Прием
● пациент_прием some Прием
● пациент_у_специалиста only Специализация

**Рис. 2.9. в.** Описание базовых классов: а) *Больничный\_лист* с помощью необходимых условий; б) *Прием* с помощью необходимых условий; в) *Пациент* с помощью необходимых условий

**Шаг 6.** На вкладке *Individuals* задаются индивидуальности, т.е. экземпляры классов (рис. 2.10). Добавлять их можно как в класс *Thing*, так и каждую индивидуальность в свой класс. Для этого необходимо нажать клавишу и ввести имя индивидуальности.

Далее для каждой индивидуальности в секциях *Object Properties* (+) и *DataProperties* (+) определяются ее связи с характеристиками и с другими объектами предметной области, т.е. задаются экземпляры отношений, например: *Орлов\_Г.П.* был на приеме у врача *Терентьева\_A.П.*, дата приема *08.06.2013* и получил больничный лист с номером *1*. На рис. 2.11 приведены экземпляры отношений (свойства) индивидуальности прием *8.06.2013*. Следует отметить, что экземпляры отношений связывают между собой не классы, а индивидуальности. Рис. 2.11, б иллюстрирует ввод экземпляра отношения *прием\_врач Терентьев\_A.P.* Если при вводе экземпляра отношения отсутствует в *Thing* необходимая индивидуальность (например, *Терентьев\_A.P.*), то ее в этот момент можно добавить, нажав кнопку .

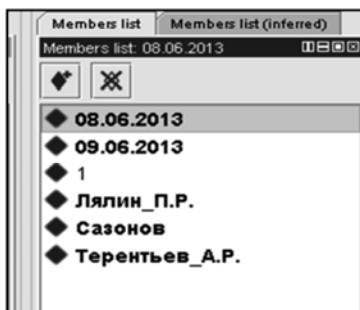


Рис. 2.10. Задание экземпляров (индивидуальностей) класса *Thing*

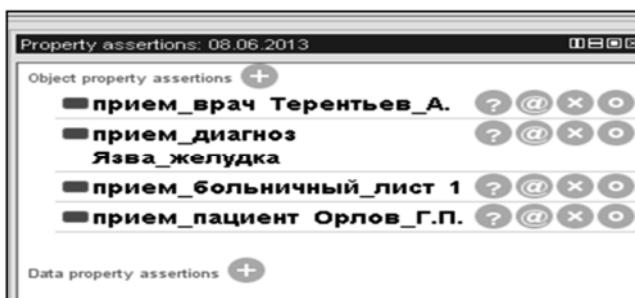
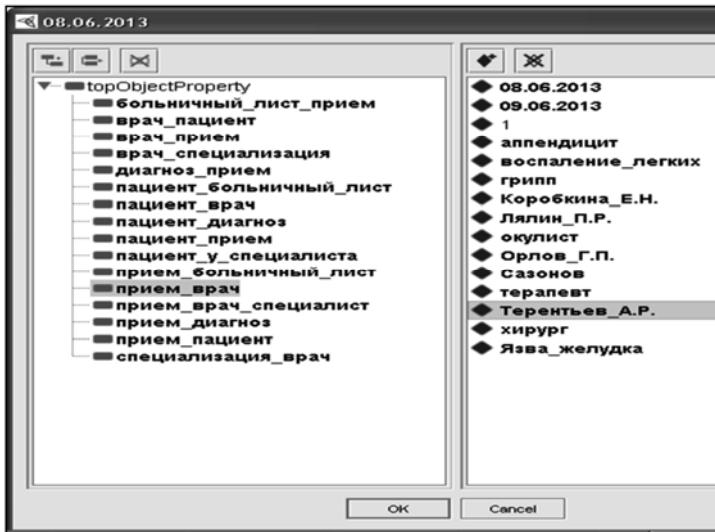


Рис. 2.11 а. Определение свойств индивидуальности прием *8.06.2013*



**Рис. 2.11 б.** Определение бинарного отношения  
08.06.2013 прием\_врач Терентьев\_А.Р.

После того как будут описаны свойства индивидуальностей в виде *Object Properties* и *DataProperties*, можно применить машину вывода, чтобы получить дополнительную информацию.

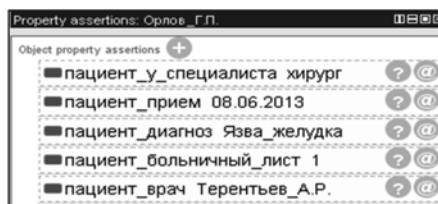
**Шаг 7.** Резонер – это машина вывода, которая, основываясь на представлении онтологии в виде логических формул, осуществляет вывод ответа на запрос пользователя, проверку непротиворечивости логических описаний и корректность классификации объектов предметной области, а именно: правильность отношений класс – подкласс в разработанной онтологии.

Рассмотрим, что будет происходить при применении машины вывода (*Резонера*) на примере. Добавим в класс *Thing* информацию о приеме на дату 8.06.2013 пациента Орлова Г.П. врачом Терентьевым А.Р., во время которого врач поставил диагноз «Язва желудка» и выдал больничный лист (рис. 2.11, а). Опишем свойства приема следующим образом: 8.06.2013 прием\_пациент Орлов\_Г.П.; 8.06.2013 прием\_диагноз Язва\_желудка; 8.06.2013 прием\_врач Терентьев\_А.И., 8.06.20 прием\_больничный\_лист 1.

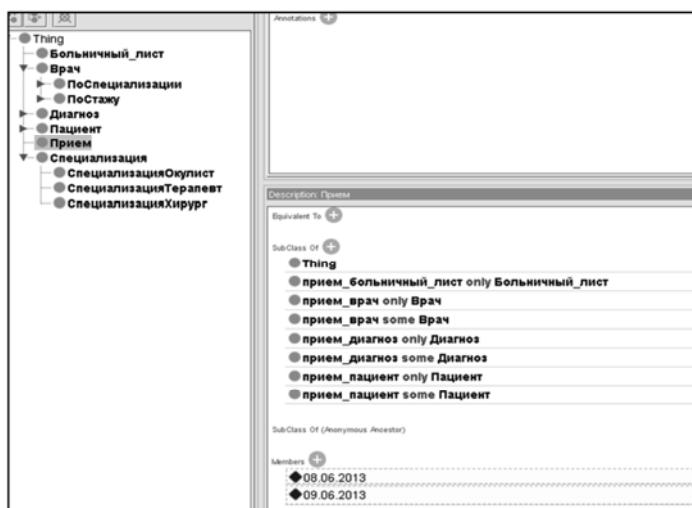
Запустим *Резонер*. В результате машина вывода получит дополнительную информацию об индивидуальностях, которые мы

внесли в онтологию. Так, например, для пациента *Орлова\_Г.П.* получена информация, представленная на рис. 2.12, она выделена серым цветом.

Перейдем на вкладку *Classes* и посмотрим, распределены ли введенные индивидуальности по классам предметной области. Однако, как мы видим, все вновь добавленные индивидуальности не отнесены ни к одной группе. Попробуем добавить информацию о том, что Орлов Г.П. – пациент (или что Терентьев – врач) и вновь повторим вывод с помощью резонера. Теперь (вкладка *Classes*) все индивидуальности правильно распределены по классам: Орлов Г.П. – это пациент, Терентьев А.Р. – это врач, Язва желудка – это диагноз, 1 – Больничный лист. На рис. 2.13 показаны элементы класса *Прием*, полученные путем вывода.



**Рис. 2.12.** Свойства индивидуальности *Орлов Г.П.*, полученные путем вывода



**Рис. 2.13.** Резонер все приемы поместил в класс *Прием*

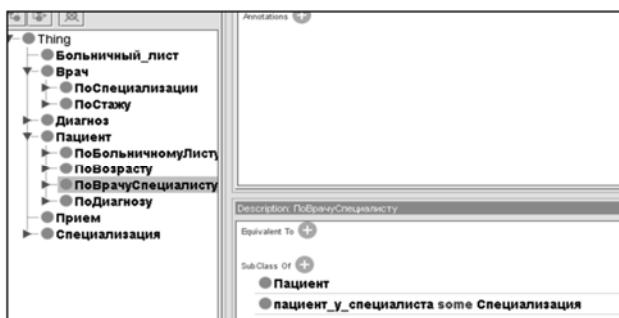
**Шаг 8.** Используя введенные отношения и описание классов с помощью логических формул, можно элементы каждого класса разбить на группы (подклассы).

Рассмотрим особенности разбиения классов на подклассы. Обычно классификация выполняется по одному или нескольким признакам (свойствам). Один и тот же класс может быть разбит на группы различным образом в зависимости от выбранного для классификации критерия. Поэтому внутри класса удобно создать подклассы: *<Классификация\_по\_первому\_признаку>*, *<Классификация\_по\_второму\_признаку>*, *<Классификация\_по\_третьему\_признаку>* и т.д. Каждый из них описать как подкласс *примитивного анонимного класса* (т.е. в секции *SubClasses Of*), указав ту характеристику (роль или связь), по которой осуществляется деление на подклассы. Например, для разделения пациентов по возрасту надо создать подкласс *ПоВозрасту* класса *Пациент* и описать его в секции *SubClasses Of* как подкласс анонимного класса, заданного бинарным отношением, отвечающим за связь пациентов с возрастом, аналогично для классификации по диагнозам (рис. 2.14).

*ПоВозрасту* → *Пациент and пациент\_возраст some integer*

*ПоДиагнозу* → *Пациент and пациент\_диагноз some Диагноз*

*ПоВрачуСпециалисту* → *Пациент and пациент\_у\_специалиста some Специализация*



**Рис. 2.14.** Описание подкласса *ПоВрачуСпециалисту* класса *Пациент*

Для каждого признака (Возраст, Диагноз, Специализация) создать свой класс, если он еще не создан, и в нем определить подклассы для каждого возможного значения. Например (рис. 2.15), в классе *Диагноз* надо создать подклассы по названию диагноза:

*Аппендицит, ЯзваЖелудка, ВоспалениеЛегких*, в классе возраст подклассы *Молодые, Пожилые, СреднихЛет*, в классе *Специализация* подклассы *СпециализацияОкулист, СпециализацияТерапевт, СпециализацияХирург*. Наполнить эти подклассы соответствующими индивидуальностями: *Аппендицит*  $\equiv \{\text{аппендицит}\}$ , *ЯзваЖелудка*  $\equiv \{\text{язва\_желудка}\}$ , *СпециализацияХирург*  $\equiv \{\text{хирург}\}$  и т.п.

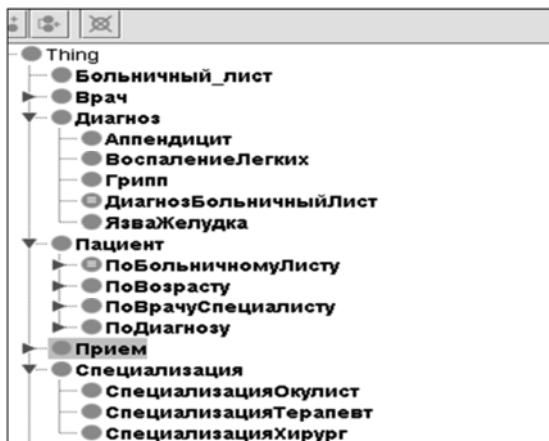


Рис. 2.15. Признаки *Диагноз, Специализация* и их подклассы

Затем внутри классификации по признаку создать подклассы, соответствующие определенным значениям признака (рис. 2.16), и их уже описать логическими формулами в разделе необходимых и достаточных условий (*Equivalent To*). Так, например, пациентов можно классифицировать по возрасту, по диагнозу, по специалистам, у которых пациент побывал на приеме (см. рис. 2.16). На рис. 2.17 приведено описание определяемого (в секции *Equivalent To*) подкласса *СреднихЛет* класса пациентов *ПоВозрасту*.

Логические формулы, описывающие определяемые (в секции *Equivalent To*) подклассы, будут иметь вид:

*ПациентыСЯзвой Желудка*  $\equiv$  *пациент\_диагноз some ЯзваЖелудка*

*ПолучилиБольничныйЛист*  $\equiv$  *(пациент\_больничный\_лист some Больничный\_лист) and (пациент\_больничный\_лист only Больничный\_лист)*

*ПациентыХирурга*  $\equiv$  *пациент\_у\_специалиста some СпециализацияХирург*

В этих формулах указывается более конкретная информация о связях элементов определяемого класса с элементами другого класса – класса-значения признака.



Рис. 2.16. Классификация пациентов по различным признакам

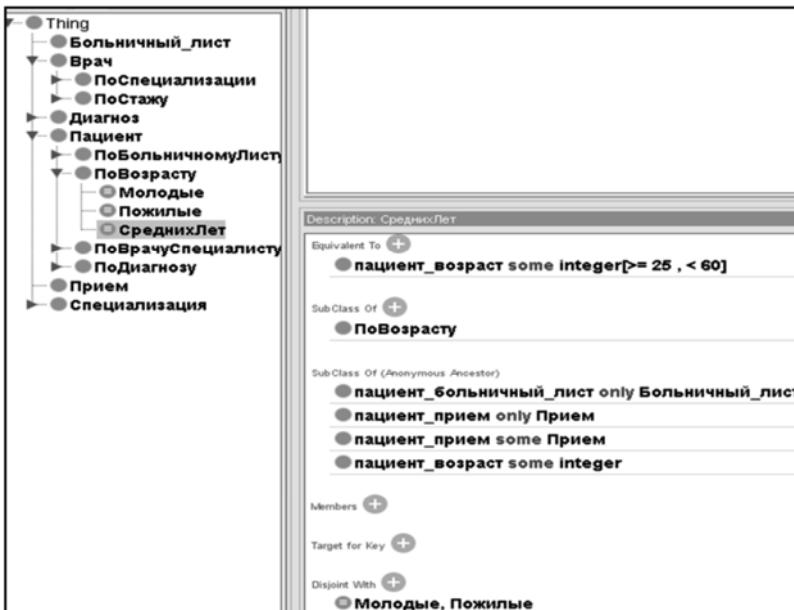


Рис. 2.17. Описание подкласса *СреднихЛет* класса *ПоВозрасту*

Аналогично можно классифицировать врачей по стажу работы, по специализации, по количеству приемов; приемы по специалистам, по выданным больничным листам; диагнозы по группам и т.д.

Например, класс врачи можно разбить на подклассы: врачи-терапевты, врачи-хирурги. Для этого надо создать в классе *Специализация* подклассы *СпециализацияТерапевт*, *СпециализацияХирург*, *СпециализацияОкулист* и объявить их непересекающимися множествами. Затем в каждый класс добавить индивидуальность: индивидуальность терапевт в класс *СпециализацияТерапевт*, индивидуальность хирург в класс *СпециализацияХирург* и т.п. Класс *Врач* представить как иерархию узлов, приведенную на рис. 2.18.

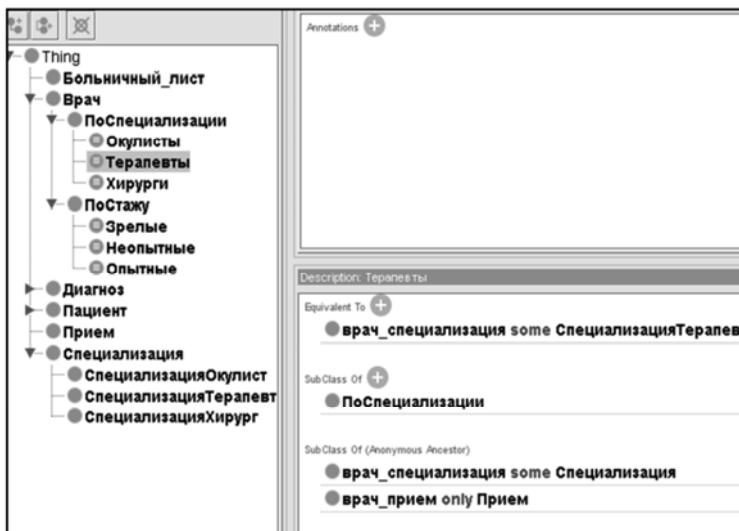


Рис. 2.18. Классификация врачей по специализации

Затем каждый из подклассов *Хирурги*, *Терапевты*, *Окулисты* описать в секции *Equivalent To* следующей формулой:

*Терапевты* = *врач\_специализация some СпециализацияТерапевт*;

*Хирурги* ≡ *врач\_специализация some СпециализацияХирург*;

*Окулисты* ≡ *врач\_специализация some СпециализацияОкулист*.

Описание класса или подкласса логическими формулами можно рассматривать как описание запроса на получение информации

из онтологии. Следовательно, для любого по сложности запроса можно создать свой поименованный класс или подкласс.

Если мы захотим узнать, при каких диагнозах были выданы больничные, то следует ввести подкласс *ДиагнозыБольничныеЛисты* класса *Диагноз*:

*ДиагнозыБольничныеЛисты*  $\equiv$  *диагноз\_больничный\_лист some БольничныйЛист*.

В дополнение рассмотрим, как описывать классы, используя *HasValue* ограничения. Для этого создадим группы пациентов, побывавших на приеме у каждого из врачей (рис. 2.19), и опишем их следующими логическими формулами:

*ПациентыЛялина*  $\equiv$  *пациент\_врач value Лялин\_П.Р.*;

*ПациентыСазонова*  $\equiv$  *пациент\_врач value Сазонов*;

*ПациентыТерентьева*  $\equiv$  *пациент\_врач value Терентьев\_А.Р.*

Запустим Резонер (машину вывода) и получим результат, показанный на рис. 2.19. Пациент Орлов Г.П. был на приеме у врача Терентьева А.Р.

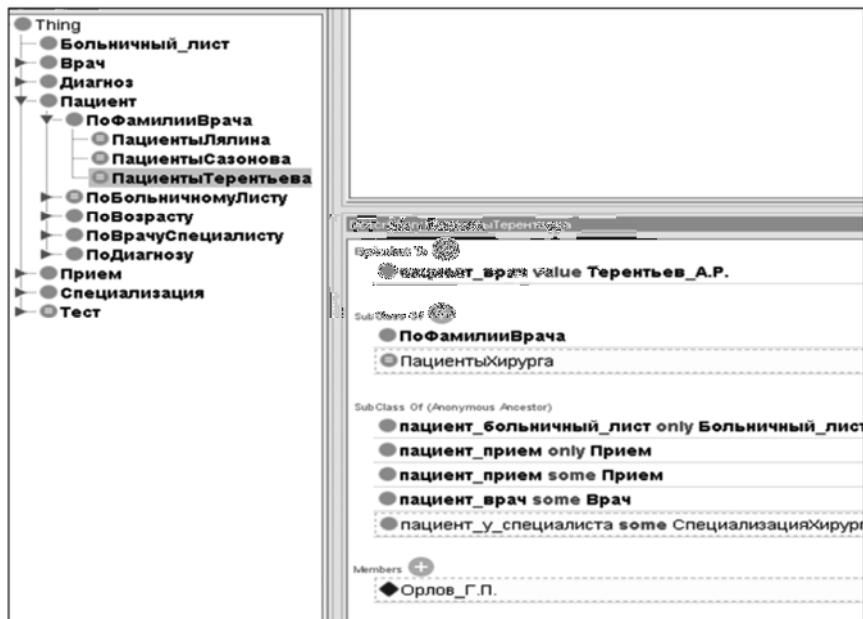


Рис. 2.19. Подклассы пациентов для каждого из врачей

**Шаг 9.** Запустим машину вывода (*Резонер*) и проверим, как она осуществляет классификацию введенных нами объектов предметной области.

Выводимая иерархия совпала с построенной при проектировании. Все подклассы заполнились соответствующими индивидуальностями: так *Орлов\_Г.П.* в соответствии с введенной о нем информацией попал в подклассы *ПациентыСЯзвойЖелудка*, *ПациентыХирурга*, *СреднихЛет* (его возраст 45 лет), *Терентьев\_А.Р.* классифицирован как *Хирурги* и по стажу как *Зрелые*, его стаж 20 лет.

Используя рассмотренную технологию, можно вести разработку онтологии предметной области по вполне определенным правилам.

С каждым элементом онтологии связан *URI* – уникальный идентификатор ресурса, однозначно задающий элемент онтологии.

Разработанная онтология может быть преобразована с помощью редактора Protege 4 в один из форматов *Semantic Web*: *RDF/XML*, *OWL* и в дальнейшем использоваться при информационном поиске в сети Интернет [20, 23].

### **Контрольные вопросы**

1. Что описывает диаграмма «сущность-связь»? Каковы правила ее построения?
2. Как создаются дерево базовых классов и дерево базовых отношений?
3. Какие производные отношения и каким образом можно создать в редакторе?
4. Как описать с помощью логических формул свойства классов, используя ограничения на связи между элементами классов?
5. Как задать ограничения на значения типизированных характеристик (атрибутов) элементов класса?
6. Каковы правила разбиения класса на подклассы по различным критериям?
7. Что такое индивидуальность? Как ее создать? Как задать характеристики и свойства индивидуальности?
8. Что такое онтограф? Как его получить?
9. Как создавать запросы к онтологии?
10. Для чего предназначен резонер? Какие функции он выполняет?

***Контрольные задания***

Разработайте онтологии для следующих предметных областей:

1. Дисциплин учебного плана вашей специальности.
2. Университета.
3. Курсовых проектов по специальности «Программная инженерия».
4. Информации, хранящейся на вашем персональном компьютере.
5. Программных средств, обеспечивающих учебный процесс по вашей специальности.
6. Стран мира.
7. Деревьев.
8. Танков.
9. Ноутбуков и нетбуков.
10. Лекарственных средств.
11. Языков Semantic Web.
12. Форм собственности.
13. Компьютеров и/или их составляющих.
14. Социальных сетей.

*Математики как французы: все переводят на свой язык, и это тотчас же становится чем-то совершенно иным.*

*Иоганн Вольфганг Гете*

## Глава 3

### ДЕСКРИПЦИОННАЯ ЛОГИКА

#### Введение

Термин «онтология» впервые появился в работе Томаса Грубера [36], в которой рассматривались различные аспекты взаимодействия интеллектуальных систем между собой и с человеком. Интеллектуальными системами называются программы, которые моделируют интеллектуальную деятельность человека. Для решения поставленных задач они используют различные знания, хранимые и накапливаемые в базах знаний.

Знания, которые заложены в компьютерных программах, можно разделить на процедурные и декларативные.

Процедурные знания – знания о том, что надо сделать в каждой конкретной ситуации, задаются с помощью алгоритма.

Декларативные знания – знания о мире задачи, т.е. знания, описывающие свойства предметной области, в которой решается поставленная задача.

Идея Грубера состояла в том, чтобы позволить интеллектуальным системам обмениваться между собой заложенными в них знаниями о мирах задач. Если внутри интеллектуальной системы знания о мире могут быть закодированы как угодно, то для обмена этими знаниями с другой интеллектуальной системой необходимо предоставить описание этих знаний на общем для систем языке. Это описание должно быть в достаточной степени формальным, чтобы понимание всеми общающимися системами (в том числе и человеком) было однозначным, а также должен быть известен язык этого описания.

С давних пор считается естественным представлять знания о предметной области в виде иерархии структурированных объектов, связанных между собой отношениями. На этой идеи базируются такие формализмы, как фреймы, семантические сети, *UML* и т.д. К сожалению, все эти языки лишены формальной семантики, т.к. выражаемая в них информация предназначена для человеческого, а

не машинного восприятия. Машине надо объяснять формально, что один класс вложен в другой класс, или, что свойства класса-предка наследуются классом-потомком.

Такое описание можно было бы сделать с использованием логики предикатов первого порядка. Но с ней существует ряд проблем: нет удобной поддержки иерархий, нет удобных средств описания структурированных классов, а также логический вывод разрешим только наполовину.

Таким образом, для представления знаний в базах знаний параллельно развивались и использовались такие формализмы как:

- семантические сети и фреймы, где было все, кроме формальной семантики [7, 16, 20];

- логика предикатов, в которой была семантика, но не было удобных средств для организации знаний [27, 35, 39].

Для общения интеллектуальных систем между собой Груббер предложил объединить два способа описания знаний:

- в канонической форме, которая представляет собой описание знаний на языке логики предикатов [27] (например, на языке *Prolog*);

- в форме онтологии, которая представляет собой множество классов (понятий), связанных между собой отношением обобщения (т.е. обратным отношением для отношения наследования).

Таким образом, онтология – это представление декларативных знаний, сделанное в виде классов с отношением иерархии между ними. К этому описанию, предназначенному для чтения человеком, присоединено описание в канонической (логической) форме, которое предназначено для чтения машинами. Грубер считал, что интеллектуальные системы будут свободно обмениваться онтологиями между собой. Современные языки описания онтологий позволяют совместить каноническую форму и форму в виде дерева классов в единое целое.

Объединение лучших элементов из двух способов описания знаний породило новый раздел в математической логике – называемый дескрипционной логикой.

Так как в основе языков описания онтологий лежит некоторая логическая (формальная) система [27], то вначале рассмотрим формальные системы (дескрипционные логики), лежащие в основе этих языков, а затем и сами языки описания онтологий.

Базовыми источниками,ложенными в основу данного раздела, являются: Борис Конев. Онтология и представление знаний: <http://www.lektorium.tv/lecture/?id=13064>, Дескрипционная логика (лекции) с.н.с. Евгений Золин: <http://lpcs.math.msu.su/~zolin/dl/>, Цуканова, Н.И. Теория и практика логического программирования на языке Visual Prolog 7. / Н.И.Цуканова, Т.А. Дмитриева // Учебное пособие для вузов. – М.: Горячая линия – Телеком, 2011. – 232.

## 1.1. Общие сведения

**Описательные логики или дескрипционные логики** (сокр. ДЛ, *description logics*) – семейство языков представления знаний, позволяющих описывать понятия предметной области в недвусмысленном, формализованном виде [60]. Они сочетают в себе, с одной стороны, богатые выразительные возможности, а с другой – хорошие вычислительные свойства, такие как разрешимость и относительно невысокая вычислительная сложность основных логических проблем, что делает возможным их применение на практике. Таким образом, ДЛ представляют собой компромисс между выразительностью и разрешимостью. ДЛ можно рассматривать как разрешимые фрагменты логики предикатов [60, 27].

Свое современное название ДЛ получили в 1980-х. Прежние названия (в хронологическом порядке): *терминологические системы, логики концептов*. Изначально ДЛ зародились как расширение фреймовых структур и семантических сетей механизмами формальной логики. В настоящее время ДЛ являются важным элементом в концепции Семантической паутины, где их предполагается использовать при построении онтологий. Фрагменты *OWL-DL* и *OWL-Lite* языка веб-онтологий *OWL* также основаны на ДЛ.

Дескрипционные логики оперируют понятиями **концепт** и **роль**, соответствующими в других разделах математической логики понятиям «одноместный предикат» (или множество, класс) и «двуместный предикат» (или бинарное отношение). Интуитивно, концепты используются для описания классов некоторых объектов, например, «Люди», «Женщины», «Квартиры». Роли используются для описания двуместных отношений между объектами, например, на множестве людей имеется двуместное отношение «X есть\_родитель\_для Y», а между людьми и квартирами имеется двуместное

отношение «X имеет\_в\_собственности Y», где в качестве X и Y можно подставлять произвольные предметы.

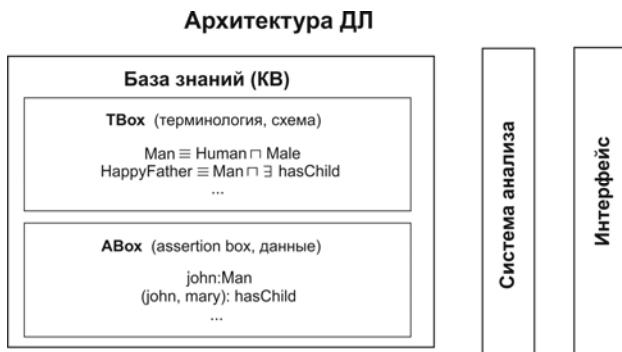


Рис. 3.1. Архитектура ДЛ

С помощью языка ДЛ можно формулировать утверждения общего вида – о классах вообще (всякая Женщина есть Человек, всякая Квартира имеется\_в\_собственности хотя бы у одного Человека) и частного вида – о конкретных объектах (Ирина есть Женщина, Владимир имеет в собственности Квартиру 52).

На жаргоне ДЛ набор утверждений общего вида или терминологии (*terminology*) называется *TBox*, набор утверждений (*assertions*) частного вида – *ABox*, а вместе они составляют так называемую базу знаний или онтологию (рис. 3.1).

Проиллюстрируем введенные понятия на примере маленькой онтологии «Семья», созданной в редакторе *Protege 4* и приведенной на рис. 3.2.

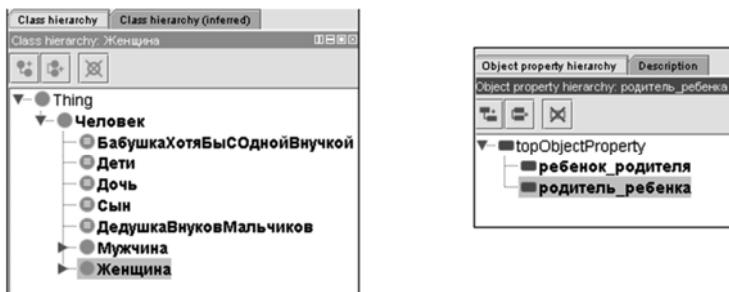


Рис. 3.2. Концепты (Классы слева) и роли (бинарные отношения справа) в онтологии «Семья»

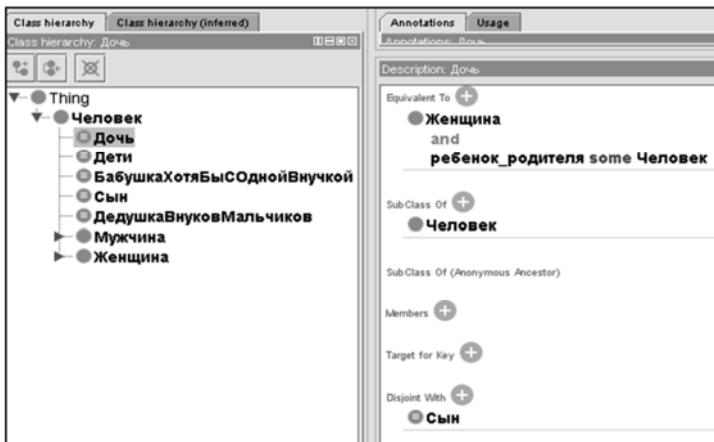


Рис. 3.3. Описание концепта *Дочь* с помощью аксиомы

На вкладке *Classes* введены концепты *Человек*, *Мужчина*, *Женщина*, *Сын*, *Дочь*, *Дети* и т.д. На вкладке *Object property* определены бинарные отношения *родитель\_ребенок* и инверсное ему отношение *ребенок\_родитель*. Каждый концепт на вкладке *Classes* может быть описан логической формулой (аксиомой) и это описание определяет терминологию *TBox* базы знаний (рис. 3.3).

На вкладке *Individuals* задаются элементы введенных классов и определяются их свойства. Эти элементы и составляют вторую часть базы знаний – *ABox* (рис. 3.4).

Многочисленные онтологии построены и строятся в самых различных предметных областях, таких как биоинформатика, генетика, медицина, химия, биология. Как только онтология построена, встает вопрос о том, как можно извлекать знания, следующие из содержащихся в онтологии знаний, можно ли это делать программно и каковы соответствующие алгоритмы. Все эти вопросы решаются теоретически в науке «дескрипционная логика», а практически уже реализовано множество программных систем – механизмов рассуждений (*reasoners*), которые позволяют автоматизировано выводить знания из онтологий и производить другие операции с онтологиями.

В качестве примера на рис. 3.5, 3.6 приведены результаты работы машины вывода (*reasoner*), с помощью которой извлечены новые знания, выделенные серым цветом. На вкладке *Class hierarchy (inferred)* показана полученная путем вывода иерархия

объектов. Она задает отношение вложенности (подкласс – класс) между всеми концептами предметной области.

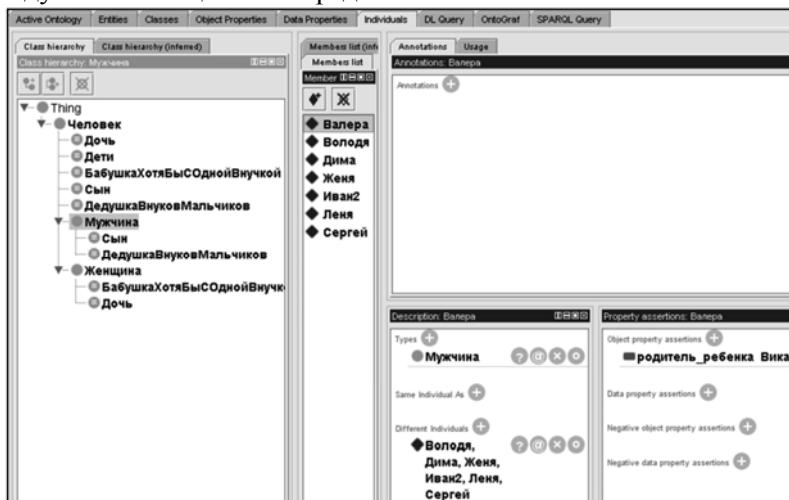


Рис. 3.4. Определение свойств экземпляров класса *Мужчина* – *Abox*

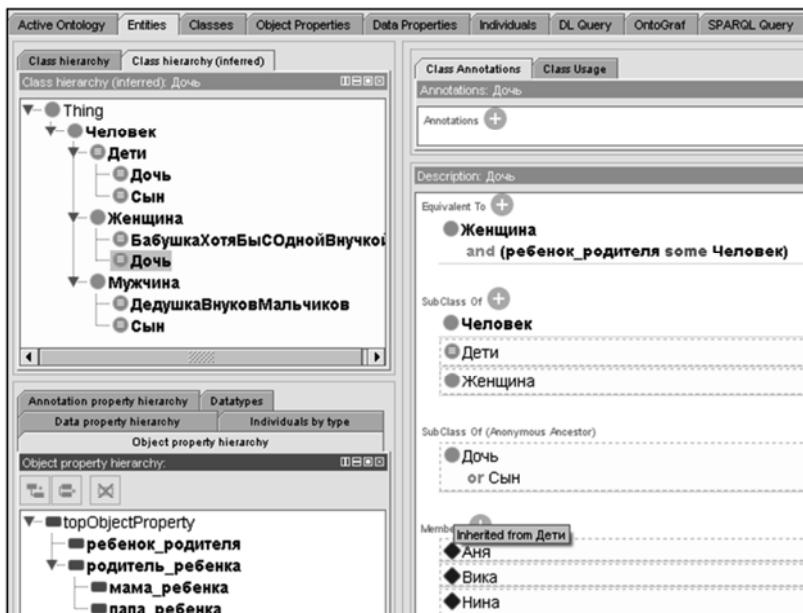


Рис. 3.5. Результаты работы машины вывода



Рис. 3.6. Характеристики индивидуальности *Вика*, заданные (черный цвет) и полученные путем вывода (серый цвет)

В правой нижней части окна в разделе *Members* путем вывода получены индивидуальности (*Аня*, *Вика*, *Нина*) класса *Дочь*. Если левой кнопкой мыши щелкнуть по одной из них, появятся характеристики соответствующей индивидуальности (рис. 3.6).

Черным цветом окрашены характеристики, заданные разработчиком онтологии, а серым цветом – выведенные *reasoner*. Так для индивидуальности *Вика* получаем, что она является дочерью, ребенком родителя *Валера* и ребенком родителя *Нина*.

Таким образом, из примеров следует, что машина вывода может быть использована для вывода иерархии классов предметной области (ПО), для распределения индивидуальностей на основе их свойств по классам, подклассам, подгруппам, для определения новых свойств, явно не заданных разработчиком, элементов ПО.

Наша задача познакомиться с теорией, лежащей в основе механизма вывода, и на простых примерах понять, как это происходит.

### 3.2. Синтаксис

В математической логике всякий язык характеризуется своим синтаксисом, то есть правилами построения выражений этого языка, и семантикой, то есть способом приписывания этим выражениям некоторого формального значения, например, указанием, какие выражения считаются истинными и ложными, какие объекты предметной области соответствуют формуле языка.

Чтобы сформулировать синтаксис какой-либо ДЛ, необходимо задать непустые (и обычно конечные) множества символов ( $C$ ,  $D$ ,  $A$ ,  $R$ ,  $R_1$  и т.д.) – так называемых *атомарных концептов* и

атомарных ролей – из которых будут строиться выражения языка данной логики. Конкретная ДЛ характеризуется набором *конструкторов* и индуктивным правилом, с помощью которого составные концепты данной логики строятся из атомарных концептов и атомарных ролей, используя эти конструкторы.

Так как язык логики абстрактен и малопонятен для большинства разработчиков, то в редакторе *Protege 4* для описания формул ДЛ используется так называемый *Манчестерский синтаксис*. Поэтому ниже все формулы ДЛ будем сопровождать их определением с использованием *Манчестерского синтаксиса* (выражение в скобках).

Типичными конструкторами для построения составных концептов ДЛ являются:

- пересечение (или конъюнкция) концептов, обозначается как  $C \sqcap D$  (*C and D*);
- объединение (или дизъюнкция) концептов, обозначается как  $C \sqcup D$  (*C or D*);
- дополнение (или отрицание) концепта, обозначается как  $\neg C$  (*not C*);
- ограничение на значения роли (или ограничение квантором всеобщности), обозначается как  $\forall R.C$  (*ребенок\_родитель only Человек*) (семантика: *только* человеческие дети);
- экзистенциальное ограничение (или ограничение квантором существования), обозначается как  $\exists R.C$  (*родитель\_ребенок some Женщина*) (семантика: родители, у которых хотя бы одна девочка);
- численные ограничения на значения роли, например:  $(\leq nR)$  (*родитель\_ребенок max 1 Женщина*) (семантика: родители, у которых не более одной девочки);  $(\geq nR.C)$  (*родитель\_ребенок min 2 Женщина*) (семантика: родители, у которых как минимум две девочки), и другие.

Как видим, в ДЛ конъюнкция и дизъюнкция обозначаются иначе, чтобы подчеркнуть отличие от других видов логик. Существуют дескрипционные логики, в которых имеются также *составные роли*, строящиеся из простых ролей с помощью операций: инверсии, пересечения, объединения, дополнения, композиции ролей, транзитивного замыкания и других.

### 3.3. Синтаксис логики ALC

Дескрипционная логика *ALC* (от *Attributive Language with Complement*) была введена в 1991 году и является одной из базовых ДЛ, на основе которой строятся многие другие ДЛ [30, 31, 33, 55, 60]. Пусть заданы непустые конечные множества атомарных концептов  $CN = \{A_1, \dots, A_m\}$  и атомарных ролей  $R = \{R_1, \dots, R_n\}$ . Тогда следующее является индуктивным определением *составных концептов* логики *ALC* (для краткости в этом определении будем называть их просто *концептами*):

- всякий атомарный концепт является концептом;
- выражения  $T$  (*Thing*) и  $\perp$  (*NoThing*) являются концептами;
- если  $C$  есть концепт, то его *дополнение*  $\neg C$  (*not C*) является концептом;
- если  $C$  и  $D$  есть концепты, то их *пересечение*  $C \sqcap D$  (*C and D*) и *объединение*  $C \sqcup D$  (*C or D*) являются концептами;
- если  $C$  есть концепт, а  $R$  есть роль, то выражения  $\forall R.C$  (*r only C*) и  $\exists R.C$  (*r some C*) являются концептами.

Строго говоря, *ALC* – это не одна логика, а семейство логик, где каждая логика этого семейства задается выбором конкретных множеств атомарных концептов и ролей. Это аналогично заданию сигнатуры теории первого порядка.

**Пример 1.** Если атомарными концептами являются *Человек*, *Женщина*, *Мужчина*, *Мать*, *Отец*, *Родитель*, *Бездетный*, а атомарными ролями *родитель\_ребенок*, *ребенок\_родитель*, то концептами логики *ALC*, будут выражения:

#### На языке ДП

*Женщина*  $\sqcup$  *Мужчина*  
*Мужчина*  $\sqcap$  *Родитель*  
 $\forall$ *родитель\_ребенок*. $\perp$   
 $\exists$ *родитель\_ребенок.Женщина*  
*Мужчина*  $\sqcap$   $\exists$ *родитель\_ребенок.*  
(*Человек*  $\sqcap$   $\exists$ *родитель\_ребенок.*  
*Женщина*)

#### Манчестерский синтаксис

(*Женщина or Мужчина*);  
(*Мужчина and Родитель*);  
(*родитель\_ребенок only NoThing*);  
(*родитель\_ребенок some Женщина*);  
(*Мужчина and родитель\_ребенок some (Человек and родитель\_ребенок some Женщина)*).

### 3.4. Семантика

Семантика ДЛ [55, 60] задается путем интерпретации ее атомарных концептов как множеств объектов (индивидуов), выбираемых из некоторого фиксированного универсального множества (*домена*), а атомарных ролей – как множеств пар индивидов, то есть бинарных отношений на домене.

Формально, *интерпретация I* состоит из непустого множества  $\Delta^I$  (*домена*) и интерпретирующей функции, которая сопоставляет каждому атомарному концепту  $A$  некоторое подмножество  $A^I \subseteq \Delta^I$ , а каждой атомарной роли  $R$  – некоторое подмножество  $R^I \subseteq \Delta^I \times \Delta^I$ . Если пара индивидов принадлежит интерпретации некоторой роли  $R$ , то есть  $(e, d) \in R^I$ , то говорят, что индивид  $d$  является *R-последователем* индивида  $e$ .

Далее интерпретирующая функция распространяется на составные концепты и роли. Поскольку последние в каждой ДЛ свои, то в качестве примера рассмотрим семантику для описанной выше логики *ALC*.

### 3.5. Семантика логики *ALC*

Интерпретирующая функция распространяется на составные концепты логики *ALC* по следующим правилам:

- $T$  интерпретируется как весь домен:  $T^I = \Delta^I$ ;
- $\perp$  интерпретируется как пустое множество:  $\perp^I = \emptyset$ ;
- дополнение концепта интерпретируется как дополнение множества:  $(\neg C)^I = \Delta^I / C^I$ ;
- пересечение концептов интерпретируется как пересечение множеств:  $(C \sqcap D)^I = C^I \cap D^I$ ;
- объединение концептов интерпретируется как объединение множеств:  $(C \sqcup D)^I = C^I \cup D^I$ ;
- выражение  $\forall R.C$  интерпретируется как множество тех индивидов, у которых все  $R$ -последователи принадлежат интерпретации концепта  $C$ . Формально:

$$(\forall R.C)^I = \{e \in \Delta^I \mid \forall d \in \Delta^I : (e, d) \in R^I \rightarrow d \in C^I\};$$

– выражение  $\exists R.C$  интерпретируется как множество тех индивидов, у которых имеется  $R$ -последователь, принадлежащий интерпретации концепта  $C$ . Формально:

$$(\exists R.C)^I = \{e \in \Delta^I \mid \exists d \in \Delta^I : (e, d) \in R^I \wedge d \in C^I\}.$$

**Пример 2** (Интерпретация выражений примера 1). Пусть домен интерпретации  $\Delta'$  состоит из всех людей, атомарные концепты *Человек*, *Женщина*, *Мужчина*, *Мать*, *Отец*, *Родитель*, *Бездетный* интерпретируются соответственно как множество всех людей, множество женщин, мужчин, множество матерей, отцов, родителей и множество бездетных, а роли *родитель\_ребенок*, *ребенок\_родитель* как отношения «родитель X имеет ребенка Y», «ребенок X имеет родителя Z». Тогда концепты примера 1 могут быть проинтерпретированы в предметной области следующим образом:

*Женщина*  $\sqcup$  *Мужчина* – множество всех людей (мужчин или женщин);

*Мужчина*  $\sqcap$  *Родитель* – множество отцов;

$\forall$  *родитель\_ребенок*.  $\perp$  – множество людей, у которых нет детей;

$\exists$  *родитель\_ребенок.Женщина* – родители, у которых есть дочери;

*Мужчина*  $\sqcap$   $\exists$  *родитель\_ребенок.(Человек*  $\Gamma$   $\exists$  *родитель\_ребенок.Женщина)* – дедушки, у которых есть внучки.

Следующие определения являются важными в дескрипционной логике.

**Определение 4.** Концепт  $C$  выполним, если существует такая интерпретация  $I$ , что  $C^I \neq \emptyset$ . При этом  $I$  называется моделью концепта  $C$ .

Концепты  $C$  и  $D$  называются эквивалентными (обозначение  $C \equiv D$ ), если в любой интерпретации  $I$  имеем  $C^I = D^I$ . Концепт  $C$  вложен в концепт  $D$  (обозначение  $C \sqsubseteq D$ ), если в любой интерпретации  $I$  имеем  $C^I \subseteq D^I$ . Концепты  $C$  и  $D$  называются непересекающимися, если в любой интерпретации  $I$  имеем  $C^I \cap D^I = \emptyset$ .

Рассмотрим ряд эквивалентностей и вложений, справедливых для дескрипционной логики [55, 60].

$$\exists R. \perp \equiv \perp; \quad \forall R. \top \equiv \top; \tag{1}$$

$$A \sqcap B \sqsubseteq A; \quad A \sqcap B \equiv B \sqcap A \tag{2}$$

$$\begin{aligned} \exists R.(A \sqcup B) &\equiv \exists R.A \sqcup \exists R.B & \forall R.(A \sqcap B) &\equiv \forall R.A \sqcap \forall R.B \\ \exists R.(A \sqcap B) &\sqsubseteq \exists R.A \sqcap \exists R.B & \forall R.(A \sqcup B) &\sqsupseteq \forall R.A \sqcup \forall R.B \\ \neg \exists R.C &\equiv \forall R.\neg C, & \neg \forall R.C &\equiv \exists R.\neg C \end{aligned} \quad (3) \quad (4)$$

Концепт  $\exists R.\perp$  в любой интерпретации обозначает множество элементов, у которых существует  $R$  последователь, принадлежащий пустому множеству  $\perp^1$ . Так как пустому множеству никакой элемент принадлежать не может, то значит концепт  $\exists R.\perp$  обозначает множество элементов, обладающих невыполнимым свойством, т.е. пустое множество. Отсюда следует, что имеет место эквивалентность  $\exists R.\perp \equiv \perp$ . Аналогично  $\forall R.T \equiv T$ .

Рассмотрим, как некоторые из формул (1)–(4) будут описаны с использованием Манчестерского синтаксиса.

*родитель\_ребенок some Nothing*  $\equiv$  *Nothing* ;

*родитель\_ребенок only Thing*  $\equiv$  *Thing* ;  $(I')$

*родитель\_ребенок some (Мужчина or Женина)*  $\equiv$  (*родитель\_ребенок some Мужчина*)  $\text{or}$  (*родитель\_ребенок some Женина*) ;  $(3')$

*лекцию\_читает only (Преподаватель and Мужчина)*  $\equiv$  (*лекцию\_читает only Преподаватель*)  $\text{and}$  (*лекцию\_читает only Мужчина*).  $(3')$

Запишем концепты  $\forall R.\perp$ ,  $\exists R.T$  с использованием Манчестерского синтаксиса и дадим им интерпретацию: *родитель\_ребенок only Nothing* – описывает особей, у которых нет детей, *родитель\_ребенок some Thing* – описывает всех родителей, т.е. тех особей, у которых есть хотя бы один ребенок.

### 3.6. Связь с логикой предикатов

Многие ДЛ, включая *ALC*, можно рассматривать как фрагменты логики предикатов [35, 39, 27] при «естественному» переводе концептов в предикатные формулы. Если в *ALC* имеются атомарные концепты  $A_1, \dots, A_m$  и атомарные роли  $R_1, \dots, R_n$ , то для перевода вводятся одноместные предикатные символы  $P_1, \dots, P_m$  и двуместные предикатные символы  $S_1, \dots, S_n$ , а сам перевод задается индуктивно следующим образом:

– атомарные концепты  $A_i$  переходят в формулы  $P_i(x)$ , например Мужчина переходит в *Мужчина*( $x$ ), атомарные роли  $R_i$  переходят в двуместные предикаты  $S_i$ , например *родитель\_ребенок* в предикат *Родитель\_ребенок*( $x, y$ );

– пересечение  $\Pi$ , объединение  $\sqcup$  и дополнение  $\neg$  концептов переходит в булевы связки конъюнкцию  $\wedge$  (**and**), дизъюнкцию  $\vee$  (**or**) и отрицание  $\neg$  (**not**);

– выражение  $\forall R_j.C$  переходит в  $\forall y(S_j(x, y) \Rightarrow C'(y))$   
 $(\forall y(Rодитель\_ребенок(x, y) \rightarrow Женщина(y)))$ ;

– выражение  $\exists R_j.C$  переходит в  $\exists y(S_j(x, y) \wedge C'(y))$   
 $(\exists y(Rодитель\_ребенок(x, y) \wedge Мужчина(y)))$ .

В последних двух пунктах переменная  $y$  – не встречавшаяся ранее, а  $C'$  есть перевод концепта  $C$  (который уже построен по предположению индукции).

Легко видеть, что данное преобразование согласуется с описанной выше семантикой ДЛ. В любой интерпретации, если атомарные концепты  $A_i$  и атомарные роли  $R_j$  интерпретированы так же, как соответствующие им предикаты  $P_i$  и  $S_j$ , то и всякий составной концепт интерпретируется тем же самым множеством, что и соответствующая ему при переводе предикатная формула от одной переменной.

В данном переводе можно обойтись всего двумя переменными, и таким образом ДЛ *ALC* (а также многие ее расширения) можно рассматривать как фрагменты логики предикатов с двумя переменными, которая, как известно [35, 39], разрешима. Данный перевод позволяет переносить результаты о разрешимости, вычислительной сложности, разрешающих алгоритмах и т.п. из области логики предикатов в область дескрипционных логик.

Покажем как онтологию, приведенную на рис. 3.2–3.4, можно описать как фрагмент логики предикатов.

Терминология (*TBox*):

$\forall x(Мужчина(x) \rightarrow Человек(x))$ ;

$\forall x(Женщина(x) \rightarrow Человек(x))$ ;

$\forall x(Дочь(x) \equiv Женщина(x) \wedge \exists y Ребенок\_родителя(x, y))$ ;

$\forall x(Сын(x) \equiv Мужчина(x) \wedge \exists y Ребенок\_родителя(x, y))$ ;

$\forall x(Дети(x) \equiv Дочь(x) \vee Сын(x))$ .

Экземпляры классов и отношений (*ABox*).

*Мужчина(Валера). Мужчина(Володя). Женщина(Аня). Женщина(Вика).*

*Родитель\_ребенка(Валера, Вика).*

*Ребенок\_родителя(Вика, Валера).*

Здесь *Человек(x)*, *Мужчина(x)*, *Женщина(x)*, *Сын(x)*, *Дочь(x)*, *Дети(x)* – предикаты с одним аргументом, соответствующие концептам *Человек*, *Мужчина*, *Женщина*, *Сын*, *Дочь*, *Дети*. Двуместный предикат *Ребенок\_родителя(x, y)* описывает бинарное отношение между ребенком *x* и его родителем *y*. Логические операции  $\rightarrow$  (импликация),  $\equiv$  (тождество),  $\forall$  (квантор всеобщности),  $\exists$  (квантор существования),  $\wedge$  (конъюнкция),  $\vee$  (дизъюнкция) описывают связи между предикатами (между концептами предметной области).

### 3.7. База знаний

Концепты ДЛ интересны не столько сами по себе, сколько как инструмент для записи знаний об описываемой предметной области. Эти знания подразделяются на общие знания о понятиях и их взаимосвязях (интенсиональные знания) и знания об индивидуальных объектах, их свойствах и связях с другими объектами (экстенсиональные знания). Первые более стабильны и постоянны, тогда как вторые более подвержены модификациям.

В соответствии с этим делением, записываемые с помощью языка ДЛ знания подразделяются на

- набор терминологических аксиом или *TBox Ти*
- набор утверждений об индивидах или *ABox А.*

Совокупность аксиом и утверждений вместе составляют так называемую базу знаний  $K = T \cup A$  (см. рис. 3.1). Далее по отдельности рассмотрим виды аксиом и утверждений, из которых может состоять *TBox* и *ABox*.

### 3.8. Аксиомы и *TBox*

*Аксиомой вложенности концептов* называется выражение вида  $C \sqsubseteq D$ , а *аксиомой эквивалентности концептов* – выражение вида  $C \equiv D$ , где *C* и *D* – произвольные концепты. Аналогично, *аксиомой вложенности ролей* называется выражение вида  $R \sqsubseteq S$ , а *аксиомой эквивалентности ролей* – выражение вида  $R \equiv S$ , где *R* и *S* – произвольные роли. Здесь  $\sqsubseteq$  есть символ вложенности (subsumption).

*Терминологией* или *набором терминологических аксиом* или *TBox* (от англ. *terminological box*) называется конечный набор

аксиом перечисленных видов. Иногда аксиомы для ролей выделяются в отдельный набор и называют его *иерархией ролей* или **RBox**. Помимо перечисленных видов аксиом, в терминологии могут допускаться и другие аксиомы (например, транзитивность ролей).

Семантика терминологии определяется естественным образом. Пусть дана интерпретация  $I$ . Аксиома  $C \sqsubseteq D$  выполняется в интерпретации  $I$ , если  $C^I \sqsubseteq D^I$ ; в этом случае также говорят, что  $I$  является моделью аксиомы  $C \sqsubseteq D$ . Аналогично для остальных видов аксиом. Терминология  $T$  выполняется в интерпретации  $I$ , а интерпретация  $I$  называется моделью терминологии  $T$ , если  $I$  является моделью всех входящих в  $T$  аксиом.

**Пример 3.** Следующая совокупность аксиом является терминологией (или ТВоХ) в языке логики *ALC*:

*Женина*  $\equiv$  Человек  $\sqcap$  ЖенскогоПола

*Мать*  $\equiv$  Женина  $\sqcap$  Эродитель\_ребенка.Thing

Человек  $\sqsubseteq$  Эродитель\_ребенка.Человек

Доктор  $\sqsubseteq$  Человек

Интуитивно (то есть при «естественной» интерпретации, когда концепту Человек соответствует множество всех людей, роли родитель\_ребенка соответствует отношение «некто имеет ребенка» и т.д.) эти аксиомы говорят, что быть женщиной означает в точности быть человеком и быть женского пола; быть матерью означает в точности быть женщиной и иметь ребенка; у всякого человека всякий ребенок есть тоже человек; доктор является человеком.

При работе в редакторе Protege 4 аксиомы вложенности могут быть заданы (рис. 3.7) либо явно при создании подкласса класса при помощи кнопки , либо путем описания подкласса анонимного класса, задаваемого логической формулой в разделе *SubClass Of*. На рис. 3.7 приведены оба способа. Класс Дети описан как подкласс класса Человек и как подкласс анонимного класса ребенок\_родителя *some* Человек.

Аксиомы эквивалентности задаются в разделе *Equivalent To* также с помощью логических формул, описывающих анонимный класс. Пример приведен на рис. 3.8. Класс Дочь описан как класс,

эквивалентный классу, задаваемому формулой: *Женщина and (ребенок\_родителя some Человек)*

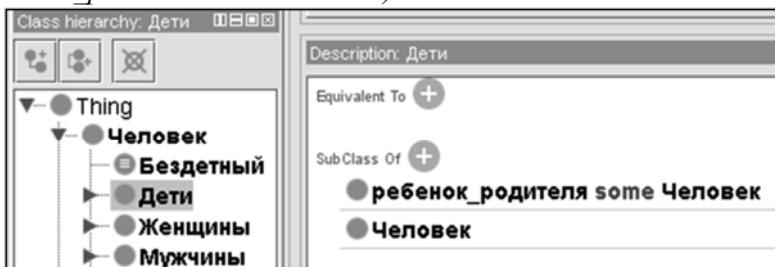


Рис. 3.7. Определение аксиомы вложенности

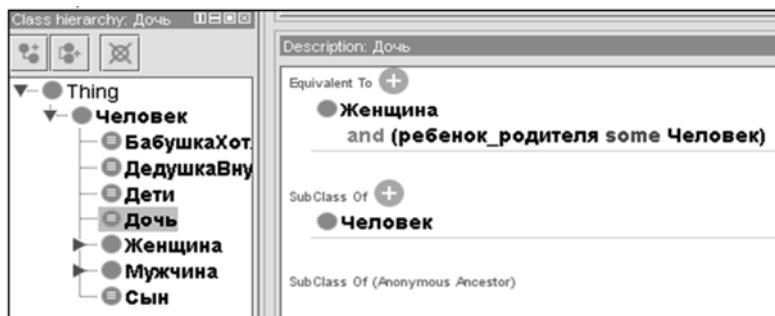


Рис. 3.8. Определение аксиомы эквивалентности

Аксиомы вложенности или эквивалентности ролей задаются на вкладке *Object Property* как показано на рис. 3.9.

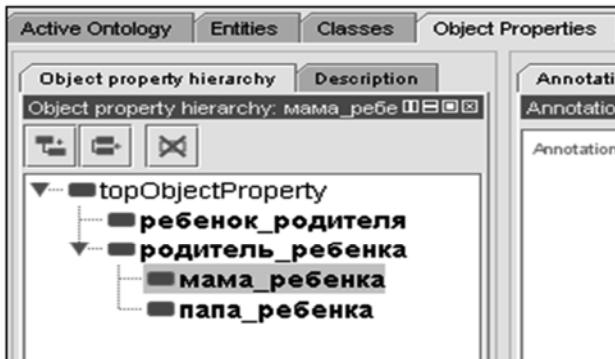


Рис. 3.9. Определение аксиомы вложенности ролей

### 3.9. Утверждения и ABox

Терминологии [60] позволяют записывать общие знания о концептах и ролях. Однако помимо этого обычно требуется также записать знания о конкретных индивидах: к какому классу (концепту) они принадлежат, какими отношениями (ролями) они связаны друг с другом. Это делается в той части базы знаний ДЛ, которая называется *ABox* (или *набор утверждений об индивидах*) (см. рис. 3.4 и рис. 3.6).

С этой целью, помимо атомарных концептов и атомарных ролей, то есть имен для классов и отношений, вводится также конечное множество имён для индивидов. Утверждения об индивидах бывают двух видов:

- утверждение о принадлежности индивида  $a$  концепту  $C$  – записывается как  $C(a)$  или  $a:C$ ;
- утверждение о связи двух индивидов  $a$  и  $b$  ролью  $R$  – записывается как  $R(a, b)$  или  $(a, b):R$  или  $a R b$ .

Наконец, *набором утверждений об индивидах* или *ABox* (от англ. assertional box) называется конечный набор утверждений этих двух видов.

**Примечание.** В некоторых ДЛ допускаются также утверждения вида  $\neg R(a, b)$  в ABox.

Чтобы задать семантику ABox, необходимо расширить интерпретацию  $I$ , а именно каждому имени индивида  $a$  сопоставить некоторый элемент домена  $a^I \in \Delta^I$ . Тогда говорят, что утверждение  $C(a)$  или  $R(a, b)$  выполняются в интерпретации  $I$ , если имеет место  $a^I \in C^I$  или  $(a^I, b^I) \in R^I$ , соответственно. Говорят, что ABox выполняется в интерпретации  $I$ , а интерпретация  $I$  является моделью данного ABox, если все его утверждения выполняются в этой интерпретации.

**Пример 4.** Следующая совокупность является набором утверждений об индивидах (или ABox) в языке логики ALC:

Галя: Женщина  $\sqcap \neg$  Доктор

Галя: Эродитель\_ребенка.Мужчина

Галя родитель\_ребенка Сергей

Сергей: Доктор  $\sqcap \forall$  родитель\_ребенка. Nothing

Здесь *Галя* и *Сергей* есть имена индивидов. Интуитивно эти утверждения означают, что *Галя* является женщиной, но не доктором, у нее есть ребенок мужского пола, *Сергей* является ребенком *Гали*, причем *Сергей* является доктором и не имеет детей.

**Примечание.** Часто рассматриваются лишь интерпретации, которые удовлетворяют *соглашению об уникальности имён (unique name assumption)*. Оно означает, что разным именам индивидов интерпретация обязана сопоставлять различные элементы домена. Язык *OWL* по умолчанию не предполагает данное соглашение, однако в нем есть конструкции, с помощью которых можно явно указать, какие имена индивидов считать равными либо различными.

На рис. 3.10 показано как в редакторе *Protege 4* на вкладке *Individuals* задаются свойства индивидуальностей. Так для *Гали* указано, что она является родителем *Сергея* и *Жени*, что она относится к типу *Женщина* и что она отличается (*Different Individuals*) от *Ани*, *Валеры*, *Вики*, *Володи* и т.д.

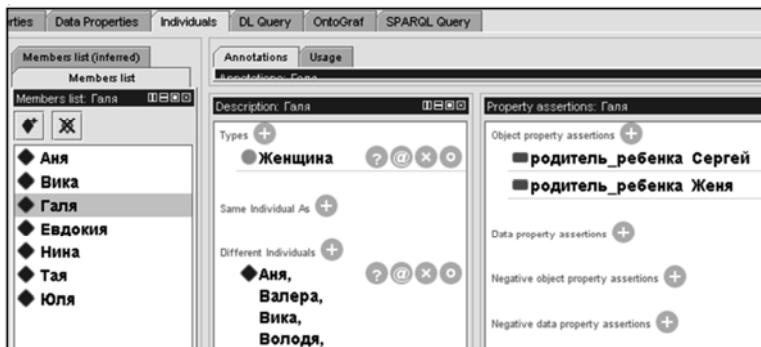


Рис. 3.10. Описание свойств индивидуальности *Галя*

### 3.10. Выразительные ДЛ

Существуют многочисленные расширения логики *ALC* дополнительными конструкторами для построения концептов, ролей, а также дополнительными видами аксиом в TBox. Имеется неформальное соглашение об именовании получающихся при этом логик – обычно путем добавления к имени логики букв, отвечающих добавленным в язык конструкторам. Наиболее известные расширения приведены в табл. 3.1. Например, логика *ALC*, расширенная ин-

версными ролями, номиналами и ограничениями кардинальности ролей, обозначается как  $ALCIOQ$ .

**Примечание.** Буква  $S$  не добавляется к имени логики, а *замещает* в нем буквы  $ALC$ . Так, например, логика  $ALC$ , расширенная инверсными ролями (буква  $I$ ), качественными ограничениями кардинальности ролей (буква  $Q$ ), транзитивными ролями (буква  $S$ ) и иерархией ролей (буква  $H$ ), имеет название  $SHIQ$ . Происхождение всех букв понятно из английских названий конструкторов; буква  $S$  означает просто *system*.

Рассматриваются также ДЛ, в которых можно строить составные роли с помощью операций объединения, пересечения, дополнения, инверсии, композиции, транзитивного замыкания и других. Кроме того, исследованы ДЛ, в которых имеются многоместные роли (обозначающие  $n$ -арные отношения).

### Таблица 3.1. Расширения логики $ALC$ дополнительными конструкторами

F	Функциональность ролей: концепты вида $(\leq 1R)$ , означающие: <i>существует не более одного R-последователя</i>
N	Ограничения кардинальности ролей: концепты вида $(\leq nR)$ , означающие: <i>существует не более n R-последователей</i>
Q	Качественные ограничения кардинальности ролей: концепты вида $(\leq nR.C)$ , означающие: <i>существует не более n R-последователей в C</i>
I	Обратные роли: если $R$ есть роль, то $R^-$ тоже является ролью, означающей обращение бинарного отношения
O	Номиналы: если $a$ есть имя индивида, то $\{a\}$ есть концепт, означающий однозначное множество
H	Иерархия ролей: в TBox допускаются аксиомы вложенности ролей $R \sqsubseteq S$
S	Транзитивные роли: в TBox допускаются аксиомы транзитивности вида $Tr(R)$
R	Составные аксиомы вложенности ролей в TBox ( $R \circ S \sqsubseteq R$ , $R \circ S \sqsubseteq S$ ) с условием ацикличности, где $R \circ S$ есть композиция ролей
(D)	Расширение языка <i>конкретными доменами</i> (типа данных)

### Контрольные вопросы

1. Какие модели представления знаний объединены в онтологиях?
2. Что такое дескрипционные логики?
3. Что такое концепт, роль, индивидуальность в дескрипционной логике?
4. Что такое ТВох (терминология) и АВох (утверждения об индивидуальностях)? Приведите примеры.
5. Что такое база знаний?
6. Опишите синтаксис логики ALC.
7. Каковы типичные конструкторы для построения составных концептов дескрипционной логики (ДЛ)?
8. Семантика логики ALC.
9. Как связаны утверждения ДЛ с логикой предикатов?
10. Какие виды аксиом Вы знаете?

### 3.11. Логический анализ

Базы знаний, описанные на языке ДЛ, применяются не только для *представления знаний* о предметной области, но также для их *логического анализа* (*reasoning*). К логическому анализу [60] относится проверка отсутствия противоречий, вывод новых знаний из уже имеющихся, обеспечение возможности делать запросы к базам знаний (по аналогии с запросами к базам данных). Благодаря тому, что базы знаний ДЛ записаны в формализованном виде, имеется возможность делать строгий логический вывод. А поскольку синтаксис и семантика ДЛ построены таким образом, что основные логические проблемы являются разрешимыми, то вывод новых знаний можно осуществлять компьютерными средствами – специальными программами (*reasoners*).

Пусть задана некоторая ДЛ. Введем несколько важных понятий.

Говорят, что концепт  $C$  данной логики *выполняется* в интерпретации  $I$ , если  $C^I \neq \emptyset$ .

Концепт  $C$  называется *выполнимым*, если существует интерпретация, в которой он выполняется.

Концепт  $C$  *вложен* в концепт  $D$  (или *содержится* в нем; англ. «is subsumed by»), если в любой интерпретации  $I$  выполняется  $C^I \subseteq D^I$ .

Аналогичные понятия можно ввести *относительно* некоторого заданного  $TBox T$ , ограничиваясь моделями данного  $TBox$ . Например, концепт  $C$  называется *выполнимым относительно*  $TBox T$ , если существует интерпретация, являющаяся моделью этого  $TBox$ , в которой данный концепт выполняется.

Когда задан не только  $TBox T$ , но и  $ABox A$ , а значит, имеется база знаний  $K = T \cup A$ , то возникает еще одно понятие.

Индивид  $a$  является *экземпляром* концепта  $C$  относительно базы знаний  $K$ , если в любой модели  $I$  базы знаний  $K$  имеет место  $a^I \in C^I$ .

Ключевыми алгоритмическими проблемами, связанными с конкретной ДЛ, являются следующие проблемы [55, 60].

**Выполнимость концепта:** является ли заданный концепт выполнимым относительно заданного  $TBox$ ?

**Вложенность концептов:** верно ли, что один заданный концепт вложен в другой относительно заданного  $TBox$ ?

**Совместимость  $TBox$ :** имеет ли заданный  $TBox$  хотя бы одну модель?

**Совместимость базы знаний:** имеет ли заданная пара ( $TBox$ ,  $ABox$ ) хотя бы одну модель?

В логиках, содержащих  $ALC$ , проблема вложенности концептов сводится к выполнимости концепта. Важное практическое значение имеют нестандартные алгоритмические проблемы, в частности:

**Классификация терминологии:** для данной терминологии (то есть  $TBox$ ) построить таксономию или *иерархию концептов*, то есть упорядочить все атомарные концепты по отношению вложения и выдать соответствующее частично упорядоченное множество.

**Извлечение экземпляров концепта:** найти все экземпляры заданного концепта относительно заданной базы знаний.

**Наиболее узкий концепт для индивида:** найти наименьший (по вложению) концепт, экземпляром которого является заданный индивид относительно заданной базы знаний.

**Ответ на запрос к базе знаний:** выдать все наборы индивидов, которые удовлетворяют заданному запросу относительно заданной базы знаний. В настоящее время глубоко изучены так называемые *конъюнктивные запросы* к базам знаний ДЛ (а также их дизъюнкции), которые похожи на аналогичные запросы из области баз данных. В случае же запросов более общего вида проблема быстро

приобретает высокую вычислительную сложность или даже становится неразрешимой.

### 3.12. Свойства ДЛ

Фундаментальными характеристиками той или иной ДЛ являются следующие [55, 60]:

*Разрешимость*: обычно рассматривают разрешимость проблем выполнимости концепта (относительно  $TBox$ ), совместимости базы знаний, ответа на конъюнктивные запросы.

*Вычислительная сложность*: изучается вычислительная сложность указанных выше алгоритмических проблем относительно размера входных данных (концепта,  $TBox$ ,  $ABox$ ). Отдельно выделяют сложность проблемы выполнимости концепта при заданном  $TBox$ , сложность проблемы выполнимости базы знаний или проблемы ответа на запросы при фиксированном  $TBox$  и меняющемся  $ABox$  (так называемая *сложность по данным*, англ. *data complexity*).

*Свойство конечности моделей*, или иначе полнота относительно конечных моделей (*finite model property*): исследуется вопрос, всегда ли верно, что если концепт выполним (относительно  $TBox$ ), то он выполним и на некоторой *конечной* модели (данного  $TBox$ ). Из наличия данного свойства у конкретной ДЛ обычно следует, что для данной ДЛ более просто строится разрешающая процедура, например, табло-алгоритм.

К настоящему времени получено большое количество результатов, касающихся этих свойств различных ДЛ. Подавляющее большинство их собрано в виде интерактивной веб-страницы: «Навигатор по сложности дескрипционных логик (англ.)» [62], где кроме того имеются ссылки на первоисточники полученных результатов.

### 3.13. Разрешимость логики $ALC$

Рассмотрим все логические проблемы, которые были перечислены в предыдущем разделе, для логики  $ALC$ . Для этого рассмотрим так называемый *табло-алгоритм* (*tableau algorithm*) [38, 55, 60], который проверяет выполнимость баз знаний в этой логике. Чтобы общее описание алгоритма было легче воспринимать, рассмотрим его работу на примере.

**Пример 5.** Пусть нужно проверить, верно ли вложение концептов:

$\exists \text{знает\_язык.Английский} \sqcap \forall \text{знает\_язык.}(\neg \text{Французский} \sqcap \text{Английский}) \sqsubseteq \exists \text{знает\_язык.Французский}$

Для этого нужно проверить на выполнимость концепт (отрицание исходного выражения  $\neg(A \rightarrow B) \equiv A \wedge \neg B$ ):

$\exists \text{знает\_язык.Английский} \sqcap \neg \forall \text{знает\_язык.}(\neg \text{Французский} \sqcap \text{Английский}) \sqcap \neg \exists \text{знает\_язык.Французский}$

Если ответ на этот вопрос будет «да», то ответ на исходный вопрос будет «нет» и наоборот (доказательство от противного).

Предварительный шаг: нормализуем концепт  $C$ , то есть продвинем все отрицания до атомарных концептов. Это можно сделать, пользуясь законами де Моргана ( $\neg(C \sqcap D) \equiv (\neg C \sqcup \neg D)$  и т.п.), законами двойственности ( $\neg \exists R.D \equiv \forall R.\neg D$  и т.п.) и законом снятия двойного отрицания ( $\neg\neg C \equiv C$ ). В результате будет получен нормализованный концепт:

$C_0 := \exists \text{знает\_язык.Английский} \sqcap \exists \text{знает\_язык.}(\text{Французский} \sqcup \neg \text{Английский}) \sqcap \forall \text{знает\_язык.} \neg \text{Французский}$

Будем строить модель, в которой этот концепт выполним (не пуст). Создадим начальную точку  $x$  с условием (0)  $x:C_0$ . Далее из этого условия будем выводить новые условия на строящуюся модель – и записывать их в протокол, имеющий вид обычного  $ABox$   $A := \{x: C_0\}$ . Поскольку  $C_0$  есть конъюнкция трех концептов, то  $x$  должен принадлежать всем трем, поэтому дописываем в  $A$  следующие факты: (1)  $x:\exists \text{знает\_язык.Английский}$ , (2)  $x:\exists \text{знает\_язык.}(\text{Французский} \sqcup \neg \text{Английский})$ , (3)  $x:\forall \text{знает\_язык.} \neg \text{Французский}$ .

Ввиду условия (1) должна существовать точка  $y$ , связанная с  $x$  отношением  $\text{знает\_язык}$ , в которой выполнен концепт  $\text{Английский}$ . Поэтому добавляем в  $A$  следующие факты: (4)  $x \text{ знает\_язык } y$ , (5)  $y:\text{Английский}$ .

Аналогично, из условия (2) следует существование точки  $z$ , связанной с  $x$  отношением  $\text{знает\_язык}$ , в которой выполнен концепт  $\text{Французский} \sqcup \neg \text{Английский}$ . Поэтому добавляем в  $A$  следующие факты: (6)  $x \text{ знает\_язык } z$ , (7)  $z: \text{Французский} \sqcup \neg \text{Английский}$ .

Далее, ввиду условия (3) во всех точках, связанных с  $x$  отношением  $\text{знает\_язык}$  (это точки  $x, z$ ), должен быть выполнен

концепт  $\neg\text{Французский}$ . Поэтому добавляем в  $A$  следующие факты: (8)  $y:\neg\text{Французский}$ , (9)  $z:\neg\text{Французский}$ .

Рассмотрим условие (7). Так как точка  $z$  принадлежит дизъюнкции (объединению) концептов, то она принадлежит одному или другому концепту. Поэтому нужно рассмотреть два случая. Первый случай: (7')  $z:\text{Французский}$  – немедленно приводит к противоречию с условием (9), согласно которому  $z:\neg\text{Французский}$ . Второй же случай: (7'')  $z:\neg\text{Английский}$ ,  $z:\neg\text{Французский}$  – никаких явных противоречий не создает.

В результате все условия доведены до элементарных, получен протокол ( $ABox A$ ), в котором никаких явных противоречий нет. Следовательно,  $ABox A$ , а вместе с ним и исходный концепт  $C_0$ , имеет модель. Фактически, сам  $ABox$  и представляет модель: нужно все созданные в процессе работы алгоритма точки считать элементами области интерпретации, а присутствующие в  $A$  элементарные факты (вида  $xRy$  или  $y:A$ ) считать заданием интерпретации атомарных концептов и ролей:

$I = (\Delta^I)$ , где  $\Delta = \{x,y,z\}$ ,  $\text{Английский}^I = \{y\}$ ,  $\text{Французский}^I = \emptyset$ ,  $\text{знает\_язык}^I = \{\langle x,y \rangle, \langle x,z \rangle\}$ .

В такой модели все факты из  $ABox$  будут верными. Следовательно, концепт  $C$  выполним, а исходное включение концептов – неверно.

Рассмотрим еще один пример уже для базы знаний «Семья», созданной в редакторе *Protege 4* и приведенной на рис. 3.2–3.6. В терминологии  $TBox$  этой базы знаний описан концепт *ДедушкаВнуковМальчиков* следующей аксиомой:

*ДедушкаВнуковМальчиков* ≡ Мужчина  
*and*(родитель\_ребенка *some*  
*(Человек*  
*and*(родитель\_ребенка *some* Мужчина)  
*and*(родитель\_ребенка *only* Мужчина))).

На языке дескрипционной логики концепт *ДедушкаВнуковМальчиков* будет описан как

Мужчина $\Gamma$ ( $\exists$ родитель\_ребенка.(Человек $\Gamma$   
 $(\exists$ родитель\_ребенка.Мужчина) $\Gamma$ ( $\forall$ родитель\_ребенка.Мужчина))).

Применим табло-алгоритм для определения выполнимости этого концепта. Концепт нормализован, в нем нет операций отри-

зания. Обозначим его кратко буквой  $\Delta$  и создадим начальную точку (0)  $x:\Delta$ . Далее по шагам, выполняя аналогичные примеру 5 действия, получим:

- (1)  $x: \text{Мужчина};$
- (2)  $x:(\exists \text{родитель\_ребенка}.(\text{Человек} \sqcap \exists \text{родитель\_ребенка.Мужчина}) \sqcap (\forall \text{родитель\_ребенка.Мужчина}));$
- (3)  $x \text{ родитель\_ребенка } y;$
- (4)  $y: \text{Человек} \sqcap (\exists \text{родитель\_ребенка.Мужчина}) \sqcap (\forall \text{родитель\_ребенка.Мужчина});$
- (5)  $y: \text{Человек};$
- (6)  $y: \exists \text{родитель\_ребенка.Мужчина};$
- (7)  $y: \forall \text{родитель\_ребенка.Мужчина};$
- (8)  $y \text{ родитель\_ребенка } z;$
- (9, 10)  $z: \text{Мужчина}.$

В полученной цепочке вывода нет противоречий. Построена база фактов  $ABox$ , которая и является моделью (интерпретацией) концепта  $\Delta$ .

$I = (\Delta, I')$ , где  $\Delta = \{x, y, z\}$ ,  $I' = \{x\}$ ,  $\text{Мужчина} = \{x, z\}$ ,  $\text{Человек} = \{y\}$ ,  $\text{родитель\_ребенка} = \{\langle x, y \rangle, \langle y, z \rangle\}$ .

Концепт  $\Delta$  не пуст, следовательно он выполним. На рис. 3.11 представлено дерево построения модели концепта  $\Delta$ .

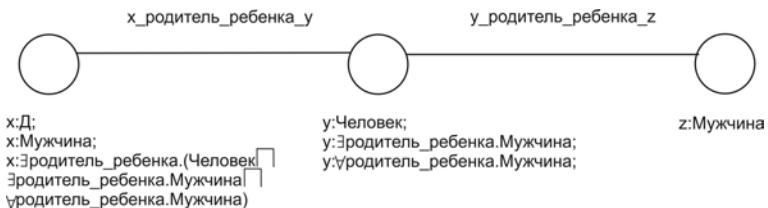


Рис. 3.11. Дерево построения модели концепта  $\Delta$

### 3.14. Понятие разрешающего алгоритма

Прежде чем описывать алгоритм и доказывать, что он «решает» нужную проблему, этому понятию дается формальное определение. Обычно в математической логике, после того, как задан язык и его семантика, пытаются построить исчисление, выводящее те и только те высказывания в этом языке, которые являются «верными» в этой семантике. Если «верность» (например, обще-

значимость) формулы  $\varphi$  обозначать как  $|=\varphi$ , а выводимость ее в исчислении обозначать как  $|-\varphi$ , то вводятся два понятия:

- исчисление называется корректным, если для любой формулы  $\varphi$  из выводимости  $|-\varphi$  следует «верность»  $|=\varphi$ ;
- исчисление называется полным, если для любой формулы  $\varphi$  из  $|=\varphi$  «верности» (общезначимости) следует  $|-\varphi$  выводимость.

Другими словами, корректное исчисление выводит только «верные» формулы. Полное – выводит все «верные» формулы.

В дескрипционной логике традиционно пытаются построить алгоритм, проверяющий, например, истинность аксиом или выполнимость концептов. Пусть зафиксирована ДЛ  $L$ , и требуется построить алгоритм  $I$ , который бы проверял вложение заданных концептов  $C \sqsubseteq D$  относительно заданной терминологии  $T$ , где  $C, D, T$  формулируются в языке  $L$ . Напомним, что это отношение обозначается как  $T |= C \sqsubseteq D$ . Ответ алгоритма  $I$  на входных данных  $C, D, T$  будем записывать как  $I(C, D, T)$ , и будем считать, что алгоритм на любом входе работает лишь конечное время и выдает только ответы 0 и 1. Тогда, по аналогии с приведенным выше определением, мы будем говорить, что алгоритм  $I$  проверки вложения концептов:

- является *корректным*, если для любых  $C, D, T$  из  $I(C, D, T) = 1$  следует  $T |= C \sqsubseteq D$ ;
- является *полным*, если для любых  $C, D, T$  и  $T |= C \sqsubseteq D$  следует  $I(C, D, T) = 1$ .

Однако, как уже говорилось ранее, в ДЛ традиционно разрабатывают алгоритмы для проверки выполнимости концептов. Как было показано ранее, вложенность концептов  $C \sqsubseteq D$  эквивалентна невыполнимости концепта  $C \sqcap \neg D$ ; соответственно, и ответы алгоритма будут противоположны тем, что при проверке вложенности концептов. Поэтому для проблемы выполнимости приходим к следующему определению разрешающей процедуры.

**Определение 2.** Алгоритм  $I$  является разрешающей процедурой для проблемы выполнимости концептов относительно терминологий для ДЛ  $L$ , если выполнены следующие три условия:

**Завершаемость:** для любых  $(C, T)$  алгоритм  $I$  выдает ответ  $I(C, T)$  через конечное время;

**Корректность:** для любых  $(C, T)$  если концепт  $C$  выполним относительно  $T$ , то  $I(C, T) = 1$ ;

**Полнота:** для любых  $(C, T)$  если  $I(C, T) = 1$ , то концепт  $C$  выполним относительно  $T$ .

### 3.15. Табло-алгоритм для логики ALC без терминологии

Пусть дан концепт  $C_0$  логики  $ALC$ , выполнимость которого требуется выяснить. Без ограничения общности [60] можно считать, что  $C_0$  уже нормализован, т.е. все встречающиеся в нем символы отрицания стоят перед атомарными концептами. Строим начальный  $ABox$   $A_0 = \{x_0 : C_0\}$ . Далее на каждом шаге к текущему  $ABox$   $A$  применяется правило – и получается один или два новых  $ABox$ . Список правил приведен в табл. 3.2. Порядок применения правил произволен (и в этом большой простор для выбора стратегий с целью оптимизации алгоритма). Правила для  $\exists, \forall, \sqcap$  из текущего  $ABox$   $A$  создают один новый  $ABox$   $A'$ , тогда как  $\sqcup$  – правило из  $ABox$   $A$  создает два новых  $ABox$   $A'$  и  $A''$ , и далее правила применяются к каждому из них «независимо».

В случае логики  $ALC$  правильность работы алгоритма не зависит от порядка применения правил. Однако это уже не так для многих других ДЛ, где требуется выбрать определенную стратегию для обеспечения завершаемости, корректности или полноты табло-алгоритма.

Таблица 3.2. Правила табло-алгоритма для логики  $ALC$

Правило	Условия применения	Действие
$\sqcap$ -правило	если 1. $x:(C \sqcap D) \in A$ 2. $x:C \notin A$ или $x:D \notin A$	то $A' := A \cup \{x : C, x : D\}$
$\sqcup$ -правило	если 1. $x:(C \sqcup D) \in A$ 2. $x:C \notin A$ или $x:D \notin A$	то $A' := A \cup \{x : C\}$ $A'' := A \cup \{x : D\}$
$\exists$ -правило	если 1. $x : \exists R.C \in A$ 2. нет такого $y$ , что $xRy \in A$ и $y : C \in A$	то создать новую точку $y$ и $A' := A \cup \{xRy, y : C\}$
$\forall$ -правило	если 1. $x : \forall R.C \in A$ 2. есть такой $y$ , что $xRy \in A$ и $y : C \notin A$	то $A' := A \cup \{y : C\}$

Таким образом, из исходного  $ABox$   $A_0$  путем применения описанных правил будет построено дерево поиска, у которого в корне наход-

дится  $A_0$  и у каждого  $ABox$  есть 0, 1 или 2 последователя. Применение правил прекращается, если к очередному  $ABox$   $A$  не применимо ни одно из правил, либо если в  $A$  содержится явное противоречие, так что применять дальнейшие правила в надежде построить модель не имеет смысла. Эти два вида  $ABox$  будут листьями дерева поиска (назовем их листовыми  $ABox$ ). Введем соответствующие термины:

- $ABox A$  называется противоречивым, если он содержит факты  $x: A \wedge x: \neg A$  для некоторого индивида  $x$  и атомарного концепта  $A$ , либо он содержит факт  $x:\perp$  для некоторого индивида  $x$ .
- $ABox A$  называется полным непротиворечивым, если он не является противоречивым, в то же время ни одно из правил табло-алгоритма к нему не применимо.

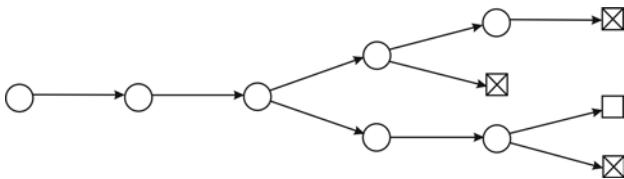


Рис. 3.12. Дерево поиска

Дерево поиска можно схематично изобразить рис. 3.12, где обозначено символом  $\circ$  – произвольный  $ABox$ ,  $\boxtimes$  – противоречивый  $ABox$ ,  $\bullet$  – полный непротиворечивый  $ABox$ .

Если алгоритм встречает полный непротиворечивый  $ABox$ , то он выдает ответ 1 и оставшиеся ветви дерева поиска уже не обходит. Напротив, если алгоритм встречает противоречивый  $ABox$ , то это лишь означает, что данная ветвь дерева поиска не привела к модели; тогда алгоритм продолжает строить остальные ветви дерева поиска. Наконец, когда алгоритм обошел всё дерево поиска, и оказалось, что все листовые  $ABox$  противоречивы, алгоритм выдает ответ 0. Итак, по построению табло-алгоритм возвращает значение 1, если хотя бы один из листовых  $ABox$  является полным непротиворечивым (и возвращает 0 в противном случае).

Формально табло-алгоритм можно описать в виде рекурсивной процедуры  $SAT(A)$ , описанной в табл. 3.4. Она принимает на вход произвольный  $ABox$   $A$  (с условием, что все концепты в нем нормализованы – а исходный  $A_0$  как раз такой) и выдает ответ 0 или 1.

Уточним, что используемый в тексте процедуры оператор  $return d$  возвращает значение  $d$  в качестве результата и прекращает

выполнение процедуры (т.е. написанные после него операторы не выполняются). В частности, выполнение приведенной процедуры дойдет до последнего оператора *return 0* только в том случае, если ни один из предыдущих операторов *return* не выполнился.

Таблица 3.3. Табло-алгоритм для логики ALC

**Функция SAT( $A$ )**

```
{ если  $A$  есть  $x: \perp$  для некоторого  $x$ , то return 0;
  если  $A$  есть  $x: A$  и  $x \neg A$  для некоторого  $x$  и  $A$ , то return 0;
  если к  $A$  не применимо ни одно из правил, то return 1;
  если применимо правило  $\Pi$ ,  $\exists$  или  $\forall$ , то применить любое, получив  $A'$ , и return SAT( $A'$ );
  если применимо  $\Box$ -правило, то применить его, получив  $A'$  и  $A''$ ,
  и далее:
    { если SAT( $A')$ =1, return 1;
      если SAT( $A''$ )=1, return 1; }
  return 0;
}
```

Каждый *ABox*  $A$  можно рассматривать как размеченный граф  $(G, L)$ . Вершинами графа  $G$  являются индивиды  $x$ , встречающиеся в  $A$ ; ребра соответствуют имеющимся в  $A$  фактам вида  $xRy$ ; меткой  $L(x)$  вершины  $x$  является множество таких концептов  $C$ , что в  $A$  имеется факт  $x:C$ , то есть  $L(x) = \{C \mid x:C \in A\}$ . На рис. 3.13 приведен граф для примера 5.

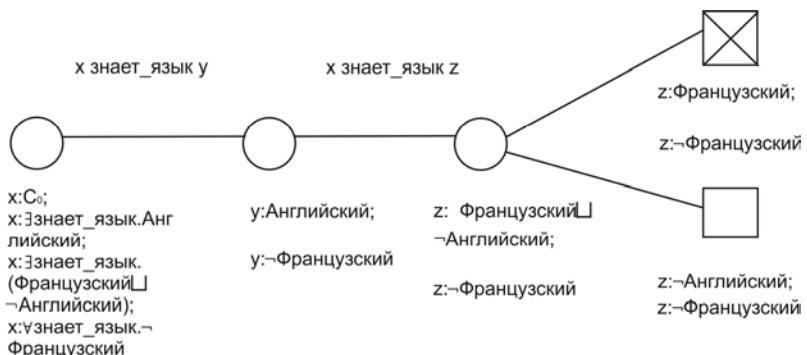


Рис. 3.13. Граф доказательства выполнимости концепта из примера 5

Для полного непротиворечивого  $ABox$  этот граф фактически представляет собой искомую модель, для поиска которой и предназначен табло-алгоритм.

Если построена модель  $I ABox A$ , то следует, что  $A$  выполним.

В дескрипционной логике [60] доказывается справедливость следующих теорем.

**Теорема 1.** (Разрешимость  $ALC$ ). Проблема выполнимости концептов логики  $ALC$  разрешима. А именно, табло-алгоритм является разрешающей процедурой для этой проблемы.

В [60] рассматривается доказательство завершаемости, корректности и полноты табло-алгоритма.

**Теорема 2.** Проблема выполнимости  $ABox$  в логике  $ALC$  разрешима.

Достаточно применить тот же табло-алгоритм, но не к  $ABox$  вида  $\{x_0: C_0\}$ , а к произвольному  $ABox A_0$ . Единственное, о чем нужно позаботиться – нормализовать все концепты в  $ABox$ . Вид графа  $G$  будет чуть сложнее: вместо дерева будет произвольный граф. Из начального  $ABox A_0$  «растут» деревья, строимые табло-алгоритмом. В оценках размера графа  $G$  и его меток нужно теперь учитывать не единственный концепт  $C_0$ , а всю совокупность концептов, встречающихся в  $ABox A_0$ .

### 3.16. Табло-алгоритм для логики $ALC$ с терминологиями

Теперь опишем алгоритм для решения более сложной проблемы: дан концепт  $C_0$  и терминология  $T$ , требуется проверить, выполним ли  $C_0$  относительно  $T$ . Без ограничения общности, концепт  $C_0$  нормализован (то есть в нем все отрицания стоят лишь перед атомарными концептами), а также  $T$  состоит из единственной аксиомы  $T \sqsubseteq E$ , где концепт  $E$  тоже нормализован.

Итак, даны концепты  $C_0$  и  $E$  и требуется выяснить, существует ли интерпретация  $I$ , в которой  $E^I = \Delta$  и  $C^I \neq \emptyset$ . Как и раньше, берем начальный  $ABox A_0 := \{x_0: C_0\}$ . На первый взгляд может показаться, что достаточно лишь добавить к описанному ранее табло-алгоритму дополнительное правило – помещающее концепт  $E$  в каждую точку  $x$  в  $ABox$ . Однако оказывается, что при этом теряется условие завершаемости. Покажем это на следующем примере.

**Пример 6.** Проверим таким способом выполнимость концепта  $A \sqcap B$  относительно терминологии  $T = \{\top \sqsubseteq \exists R.A\}$ . Стартовый  $ABox$   $A_0 = \{x_0: (A \sqcap B)\}$ . Раскрывая конъюнкцию, мы добавляем в  $ABox$  факты  $x_0: A$  и  $x_0: B$ . Добавим также факт  $x_0: \exists R.A$ .

Применим теперь  $\exists$ -правило к точке  $x_0$ , в результате чего создадим точку  $x_1$  и добавим в  $ABox$  факты  $x_0Rx_1$  и  $x_1: A$ . Добавим также факт  $x_1: \exists R.A$ . Аналогично будет создана точка  $x_2$ , такая что  $x_1Rx_2$ , с метками  $x_2: A$  и  $x_2: \exists R.A$ ; затем будет создана точка  $x_3$  и т.д. Как видим, процесс «зациклился» – он создает точки с одинаковыми метками.

Чтобы этого не происходило, нужно отслеживать подобные циклы и прерывать их. В нашем случае метка точки  $x_1$  содержится в метке точки  $x_0$ , а именно  $L(x_1) = \{A, \exists R.A\} \subseteq \{A, B, \exists R.A\} = L(x_0)$ . Это является признаком того, что дальше эту ветвь продолжать не нужно. В такой ситуации мы будем говорить, что точка  $x_1$  блокирована точкой  $x_0$ .

Заметим, что условие блокировки можно было, в принципе, сформулировать не как включение, а как равенство меток; такой алгоритм тоже будет работать правильно и всегда завершаться. Однако в этом случае он зачастую будет делать лишнюю работу, что мы и наблюдаем в нашем примере: прежде чем прервать цикл, пришлось бы построить точку  $x_2$ , метка которой в точности совпадёт с меткой  $x_1$ .

Таким образом, прежде чем формулировать табло-алгоритм, нам требуется ввести новые понятия. Пусть на очередном шаге алгоритма имеется  $ABox A$ . Меткой точки  $x$  в  $ABox A$  будем называть множество  $L(x) = \{C \mid x: C \in A\}$ . Напомним также, что в ориентированном графе вершина  $x$  называется предком вершины  $y$ , а вершина  $y$  – потомком вершины  $x$ , если из  $x$  в  $y$  ведет ориентированный путь.

**Определение 3 (Блокировка).** Точка  $x$  блокирует точку  $y$ , если  $x$  есть предок  $y$  и  $L(x) \supseteq L(y)$ .

Точку  $y$  будем называть блокированной, если она блокирована некоторой точкой  $x$ .

Блокированные точки и их потомков будем называть неактивными точками, остальные – активными.

Теперь правила табло-алгоритма сформулировать легко – нужно к каждому правилу из табл. 3.2 добавить дополнительное предусловие «точка  $x$  – активная», а также ввести новое правило, добавляющее концепт  $E$  в каждую точку. Получающаяся система правил приведена в табл. 3.4.

Сам же табло-алгоритм остается прежним (см. табл. 3.2), за исключением того, что в строчку, где фигурируют правила  $\Pi$ ,  $\exists$ ,  $\forall$ , необходимо добавить упоминание  $T$ -правила, которое, как и те правила, из  $ABox A$  создает один новый  $ABox A'$ . Построенный таким образом табло-алгоритм является разрешающей процедурой для проблемы выполнимости концептов относительно терминологий. Корректность, завершаемость, полнота алгоритма доказываются в теоретических исследованиях [30, 33, 35, 38, 55, 60].

Таблица 3.4. Правила табло-алгоритма для логики  $ALC$  с  $TBox$

Правило	Условия применения	Действие
$\Pi$ -правило	если 0. точка $x$ – активная 1. $x:(C \sqcap D) \in A$ 2. $x:C \notin A$ или $x:D \notin A$	то $A' := A \cup \{x : C, x : D\}$
$\sqcup$ -правило	если 0. точка $x$ – активная 1. $x:(C \sqcup D) \in A$ 2. $x:C \notin A$ или $x:D \notin A$	то $A' := A \cup \{x : C\}$ $A'' := A \cup \{x : D\}$
$\exists$ -правило	если 0. точка $x$ – активная 1. $x : \exists R.C \in A$ 2. нет такого $y$ , что $xRy \in A$ и $C \in A$	то создать новую точку $y$ и $A' := A \cup \{xRy, y : C\}$
$\forall$ -правило	если 0. точка $x$ – активная 1. $x : \forall R.C \in A$ 2. есть такой $y$ , что $xRy \in A$ и $y : C \notin A$	то $A' := A \cup \{y : C\}$
$T$ -правило	если 0. точка $x$ – активная 1. $x : E \notin A$	то $A' := A \cup \{y : E\}$

Выражение « $ABox A$  выполним относительно  $TBox T$ » означает то же самое, что и «база знаний  $(T, A)$  выполнима».

**Пример 7.** Рассмотрим пример доказательства вложения концептов

$Rодитель \sqsubseteq \exists родитель\_ребенка.T,$  (5)

когда база знаний описывается следующей терминологией:

$Rодитель \sqsubseteq Отец \sqcup Мать;$

*Отец* ⊑ *Мужчина* ∧ *Эродитель\_ребенка*. Т; (6)  
*Мать* ⊑ *Женщина* ∧ *Эродитель\_ребенка*. Т.

Доказательство аксиомы вложеннойности концептов (5) сводится к доказательству выполнимости концепта

*Родитель* ⊑ *Эродитель\_ребенка*. Т = *Родитель* ∧ *В родител\_ребенка*. ⊥. (7)

Представим терминологию (6) в эквивалентной дизъюнктивной форме:

Т ⊑  $\neg$ *Родитель* ∨ *Отец* ∨ *Мать*;

Т ⊑  $\neg$ *Отец* ∨ (*Мужчина* ∧ *Эродитель\_ребенка*. Т); (8)

Т ⊑  $\neg$ *Мать* ∨ (*Женщина* ∧ *Эродитель\_ребенка*. Т);

Процесс доказательства описывается следующей последовательностью шагов.

(0) x: *Родитель* ∧ *В родител\_ребенка*. ⊥;

(1) x: *Родитель*;

(2) x: *В родител\_ребенка*. ⊥;

Подключаем к доказательству первое утверждение терминологии

(3) x:  $\neg$ *Родитель* ∨ *Отец* ∨ *Мать*

На следующем шаге операция ∨ дает нам три варианта решения.

(4') x:  $\neg$ *Родитель* – получено противоречие с (1);

(4'') x: *Отец*;

(4''') x: *Мать*.

Идем по ветви (4'') и подключаем второе утверждение терминологии  $\neg$ *Отец* ∨ (*Мужчина* ∧ *Эродитель\_ребенка*. Т).

(5') x:  $\neg$ *Отец* – получено противоречие с (4'');

(5'') x: *Мужчина*;

(6'') x: *Эродитель\_ребенка*. Т.

Раскрываем (6'').

(7'') x: *родител\_ребенка* у;

(8'') y: Т, но из (2) следует, что у: ⊥, значит получено противоречие.

Идем по ветви (4''') и подключаем третье утверждение терминологии.

(9') x:  $\neg$ *Мать* ∨ (*Женщина* ∧ *Эродитель\_ребенка*. Т);

(10') x:  $\neg$ *Мать* – получено противоречие с (4'''');

(10'') x: *Женщина* ∧ *Эродитель\_ребенка*. Т;

(11")  $x: \text{Женщина};$

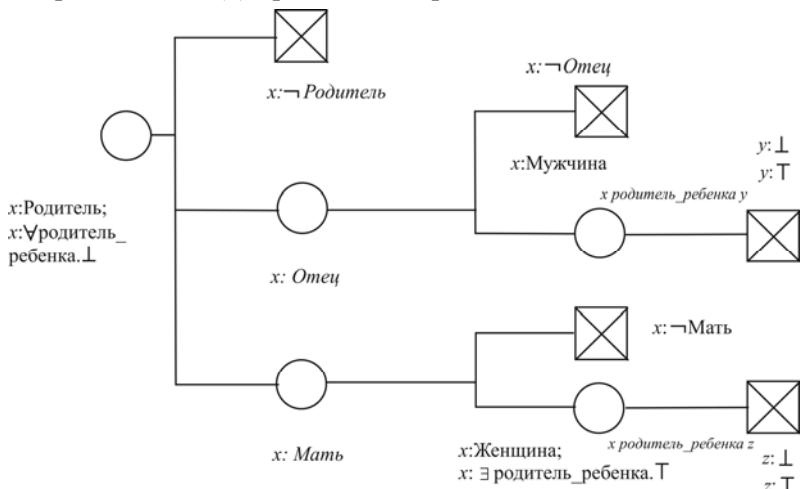
(12")  $x: \exists \text{родитель\_ребенка.} \top;$

(13")  $x \text{ родитель\_ребенка } z;$

(14")  $z: \top$ , но из (2) следует, что  $z: \perp$ , значит получено противоречие.

Таким образом, все ветви доказательства заканчиваются противоречием, следовательно, концепт (7) не выполним, а вложенность концептов (5) имеет место в данной терминологии.

Дерево доказательства вложенности концептов (5) при заданной терминологии (6) приведено на рис. 3.14.



**Рис. 3.14.** Дерево доказательства вложенности концептов при заданной терминологии

**Пример 8.** Приведем еще примеры применения табло-алгоритма к доказательству вложенности концептов. Пусть наша база знаний содержит следующие аксиомы:

$$\text{Дети} \equiv \text{Сын} \sqcup \text{Дочь} \tag{9}$$

$$\text{Сын} \equiv \text{Мужчина} \sqcap \exists \text{ребенок\_родитель. Человек} \tag{10}$$

$$\text{Дочь} \equiv \text{Женщина} \sqcap \exists \text{ребенок\_родитель. Человек} \tag{11}$$

$$\text{Отец} \equiv \text{Мужчина} \sqcap \exists \text{родитель\_ребенок. Человек} \tag{12}$$

$$\text{Мать} \equiv \text{Женщина} \sqcap \exists \text{родитель\_ребенок. Человек} \tag{13}$$

$$\text{Мужчина} \sqsubseteq \text{Человек} \tag{14}$$

$$\text{Женщина} \sqsubseteq \text{Человек} \tag{15}$$

Проверим, имеют ли место в данной терминологии следующие вложенности концептов  $\text{Сын} \sqsubseteq \text{Человек}$  и  $\text{Сын} \sqsubseteq \text{Отец}$ . Ранее было показано, что задача проверки вложенности концептов  $C \sqsubseteq D$  сводится к задаче проверки противоречивости концепта  $C \neg D$ , следовательно, в первом случае будем доказывать выполнимость концепта  $\text{Сын} \neg \text{Человек}$ , а во втором –  $\text{Сын} \neg \text{Отец}$ .

Построим модель для концепта  $C_0 := \text{Сын} \neg \text{Человек}$ . Возьмем точку (0)  $x:C_0$  и будем последовательно раскрывать (интерпретировать) наши формулы в соответствии с заданной терминологией (9)–(15). Получаем (1)  $x: \text{Сын}$ , (2)  $x: \neg \text{Человек}$ . Далее в соответствии с формулой (10) раскроем концепт  $\text{Сын}$ : (3)  $x: \text{Мужчина}$ , (4)  $x: \exists \text{ребенок\_родителя.Человек}$ . Интерпретируем описанную на шаге (4) связь: (5)  $x \text{ ребенок\_родителя } y$ , (6)  $y: \text{Человек}$ . В нашей терминологии справедлива аксиома  $\text{Мужчина} \sqsubseteq \text{Человек}$ , запишем ее в эквивалентной форме  $\neg \text{Мужчина} \sqcup \text{Человек}$ , элемент  $x$  должен удовлетворять этой аксиоме, поэтому добавим (7)  $x: \neg \text{Мужчина} \sqcup \text{Человек}$  к нашей модели. На шаге (7) появилась операция «или», которая порождает два решения (две ветви) (8')  $x: \neg \text{Мужчина}$  и (8'')  $x: \text{Человек}$ . В первой ветви противоречие между (3) и (8'), во второй между (2) и (8''). Так как во всех ветвях табло-алгоритма получено противоречие, то можно сделать вывод о противоречивости концепта  $C_0$ , следовательно, о справедливости вложения концептов  $\text{Сын} \sqsubseteq \text{Человек}$ .

Построим модель для концепта  $C_0 := \text{Сын} \neg \text{Отец}$ . Выполним аналогичным образом следующие шаги: (0)  $x:C_0$ , (1)  $x: \text{Сын}$ , (2)  $x: \neg \text{Отец}$ , раскрываем концепт  $\text{Сын}$ : (3)  $x: \text{Мужчина}$ , (4)  $x: \exists \text{ребенок\_родителя.Человек}$ . Интерпретируем описанную на шаге (4) связь: (5)  $x \text{ ребенок\_родителя } y$ , (6)  $y: \text{Человек}$ . Чтобы раскрыть концепт  $\neg \text{Отец}$  воспользуемся формулой (12) и нормализуем этот концепт, т.е. продвинем отрицание до атомарных концептов. (7)  $x: \neg (\text{Мужчина} \sqcap \exists \text{родитель\_ребенок.Человек})$ , это выражение эквивалентно (7)  $x: \neg \text{Мужчина} \sqcup \forall \text{родитель\_ребенок}. \neg \text{Человек}$ . На шаге (7) опять встретилась операция «или», в результате будут две ветви доказательства (8')  $x: \neg \text{Мужчина}$  и (8'')  $x: \forall \text{родитель\_ребенок}. \neg \text{Человек}$ . Первая ветвь дает противоречие (3) и (8'), чтобы выполнить вторую ветвь, надо найти среди существующих связь  $x \text{ родитель\_ребенок } z$  и указать  $z: \neg \text{Человек}$ , но такой связи в нашей

модели нет, следовательно, во второй ветви модель остается без изменений, и противоречий в ней нет. В результате концепт *СынП* – *Отец* выполним, а вложение концептов *Сын* ⊑ *Отец* несправедливо в нашей терминологии.

**Теорема 3** (Разрешимость *ALC* с терминологиями). Проблема выполнимости концептов относительно терминологий в логике *ALC* разрешима. А именно, табло-алгоритм является разрешающей процедурой для этой проблемы.

**Теорема 4.** Проблема совместности баз знаний в логике *ALC* разрешима.

**Замечание 2.** Что же такое «табло»? По сути, это синтаксическая структура, имитирующая некоторую модель и содержащая в том или ином виде исходные данные табло-алгоритма (концепт, который нужно проверить на выполнимость, терминологию, базу знаний и т.д.), тем самым гарантируя их выполнение в данной модели. Как мы видели выше, в случае логики *ALC* табло есть просто текущий *ABox*, который пополняется алгоритмом на каждом шаге. В конце работы алгоритма по табло можно непосредственно считать модель. В общем же случае, в табло-алгоритмах для других ДЛ возможны вариации:

- табло может не иметь вид *ABox* (например, в табло могут добавляться утверждения вида  $x = y$ );

- табло может не представлять модель буквально (например, роли в табло могут не быть транзитивными, тогда как в модели они должны были бы быть транзитивными, чтобы выполнялась аксиома транзитивности);

- табло может быть бесконечным, и в этом случае табло-алгоритм строит не само табло, а некоторую конечную «сокращенную структуру», из которой табло получается каким-либо способом (например, разверткой путей, ведущих из корня).

### 3.17. Отличие баз знаний от баз данных

Помимо того, что базы знаний описываются на несколько другом языке, нежели базы данных, их главное отличие заключается в использовании в ДЛ при логическом выводе так называемого предположения об открытости мира, тогда как в базах данных принимается предположение о замкнутости мира [50, 55, 60]. Последнее означает, что если некоторое утверждение не

является истинным, то оно принимается ложным. Предположение же об открытости мира в этом случае считает такое утверждение ни истинным, ни ложным. Это кардинальным образом влияет на то, какие факты считаются логически следующими из заданной базы знаний, а значит, и на само понятие логического следования [27] в ДЛ. Рассмотрим подробнее вопрос различия баз данных и баз знаний.

Реляционная база данных это совокупность связанных таблиц, каждая из которых задает множество экземпляров некоторого  $n$ -местного отношения (предиката). Эта совокупность таблиц описывает одну конкретную модель предметной области. Запрос к базе данных представляет собой произвольную формулу  $q$  логики предикатов, состоящую из логических операций, констант и предикатов, описывающих таблицы базы данных. Ответом на запрос  $q$  является либо булевское значение «истина» или «ложь», если в  $q$  нет свободных переменных, либо набор значений  $(a, b, c, \dots)$  переменных предиката  $q(x, y, z, \dots)$ , при которых этот предикат описывает истинное в предметной области отношение.

Первое отличие баз данных от баз знаний состоит в том, что в базах данных могут использоваться многоместные отношения (предикаты), тогда как в языке основанных на ДЛ баз знаний допустимы лишь 1-местные и 2-местные предикаты (концепты и роли). Это различие не столь существенно, так как существуют ДЛ с многоместными отношениями, которые сводятся к традиционным ДЛ с двуместными ролями.

Второе отличие – в базах знаний имеется компонент «терминология» (*TBox*). Здесь можно заметить, что в базах данных существует некоторый его аналог (называемый *схемой*), который гораздо менее выразителен, чем «терминология».

Третье отличие – в базах данных в качестве запросов допускаются произвольные формулы логики предикатов. В основанных на ДЛ базах знаний допустимы лишь запросы вида «концепт  $C$  выполним?», «концепт  $C$  вложен в концепт  $D?$ », «индивиду  $a$  принадлежит концепту  $C?$ » и т.п. Все они менее выразительны, чем произвольные формулы логики предикатов. Существуют также *конъюнктивные запросы*, однако и они представляют собой лишь некоторый фрагмент логики предикатов.

Но даже если ограничиться общим для баз данных и баз знаний языком (то есть одно – и двух – местными предикатами), отбросить терминологию баз знаний и схемы баз данных, и сузить класс запросов до самых простых, допустимых в базах знаний, то и в этом случае между базами данных и базами знаний будет иметься различие – четвертое, и самое важное – в их *семантике*. Суть этого различия в том, что в базах данных предполагается, что информация о данных задана полностью, тогда как в базах знаний считается, что информация неполна. Для описания этого различия введен специальный термин – принято говорить, что:

- в базах данных принято предположение о замкнутости мира (*closed world assumption*);
- в базах знаний принято *предположение об открытости мира* (*open world assumption*).

Формально, база данных представляет собою *одну* модель; а именно, ту, в которой каждое из отношений *R* состоит в *точности* из тех самых элементов, пар, троек и т.п., которые заданы в этой базе данных. Именно по этой причине проблема ответов на запросы является разрешимой, несмотря на то, что запросами могут быть произвольные формулы логики предикатов (которая, является неразрешимой). Дело в том, что проблема ответов на запросы к базе данных представляет собой не логический вывод или проверку логического следствия одних утверждений из других, а проверку утверждения на одной фиксированной конечной модели. Например, если в базе данных задан факт *Мария родитель\_ребенка Петя* и других фактов не дано, то это, с точки зрения базы данных, означает, что *Мария* имеет ровно одного ребенка *Петя*. Если дополнительно дан факт *Петя:Мужчина*, то на запрос *Мария: ∀ родитель\_ребенок.Мужчина* (т.е. «верно ли, что все дети Марии – мужского пола?»), будет дан ответ «да», поскольку база данных считает, что информация задана полностью и других детей у *Марии* нет.

Совершенно иначе обстоит дело с базой знаний (собственно, ровно также, как и в любой математической теории, например, в логике предикатов). База знаний описывает целое семейство моделей (а именно, всех тех интерпретаций, в которых все аксиомы и факты данной базы знаний верны). Поэтому факт *Мария родитель\_ребенка Петя*, помещённый в АBox, здесь означает

лишь, что у Марии есть, *по крайней мере*, один ребенок Петя, даже если в АВоХ никаких других фактов нет. При этом данный АВоХ имеет и модель, где у Марии ровно один ребенок, и модели, где у Марии есть и другие дети. Чтобы утверждать, что у Марии ровно один ребенок, требуется в АВоХ добавить факт Мария:( $\leq 1$  родитель\_ребенка.Thing). Это и означает, что информация в базе знаний считается заданной не полностью.

Различие в семантике БД и БЗ приводит к различиям в «способах мышления», требуемых для получения ответа на запрос. Например, для ответа на запрос  $\langle a:C? \rangle$  к БД достаточно лишь вычислить  $C^I$ , где  $I$  – та самая единственная модель, представленная этой БД, и проверить, что  $a$  принадлежит этому множеству. В БЗ же требуется именно логический вывод, при этом необходимо учесть *все* модели данной БЗ и в каждой проверить, что  $a^I \in C^I$ .

**Пример 9.** Пусть мы имеем набор фактов:  $\{a:A, b:B, aRb\}$ . На запрос  $x:\neg A$  (то есть «выдать все  $x$ , не принадлежащие  $A$ ») база данных выдаст  $\{b\}$ , а база знаний выдаст  $\emptyset$ .

На запрос  $x:\forall R.B$  («выдать все  $x$ , из которых все  $R$ -последователи ведут в  $B$ ») база данных выдаст  $\{a\}$ , а база знаний выдаст  $\emptyset$ .

Заметим, что базы знаний *монотонны*: если база расширяется новыми фактами (или аксиомами), то множество, выдаваемое в качестве ответа на запрос, тоже не сужается а, возможно, и расширяется. Напротив, базы данных не монотонны: если мы добавим к рассмотренным выше фактам новый факт  $b:A$ , то в ответ на запрос  $x:\neg A$  база данных уже выдаст  $\emptyset$ . Аналогично, добавление факта  $aRa$  приведет к тому, что в ответ на запрос  $x:\forall R.B$  база данных выдаст  $\emptyset$ .

Еще одно наблюдение состоит в том, что ответ базы знаний всегда есть подмножество ответа базы данных. Действительно, ответ базы знаний должен быть следствием из заданных фактов во *всех* моделях, тогда как ответ базы данных является следствием из этих же фактов только в *одной* из моделей.

В базах знаний из-за неполноты информации при выводе возникает необходимость перебора всех альтернатив. Такой перебор является одним из источников высокой вычислительной сложности ДЛ.

Рассмотрим примеры с базой знаний «Семья» и покажем, почему при описании концепта, казалось бы, очевидной логической формулой, не получаем ожидаемого результата. Пусть в нашей базе знаний о людях мы хотим индивиды разделить на два класса «Родители» и «Бездетные». Попытаемся сделать это с помощью следующей терминологии:

*Человек*  $\equiv$ Родитель  $\sqcup$  Бездетный

Родитель  $\sqcap$  Бездетный  $\equiv \emptyset$

Родитель  $\equiv$ Эродитель\_ребенок. Thing

Бездетный  $\equiv$ Человек  $\sqcap$  –Родитель (16)

Бездетный  $\equiv$ Человек  $\sqcap$   $\forall$  родитель\_ребенок. Nothing (17)

Если занести в *ABox* факты типа «*Нина родитель\_ребенка Вика*» о родословной некоторой семьи, то при желании получить всех родителей – мы их получим в результате вывода, а вот класс *Бездетный* будет пуст и при использовании формулы (16) и при использовании формулы (17). Такой результат можно объяснить тем, что отсутствие факта наличия ребенка для некоторой индивидуальности еще не означает, что ребенка нет, а говорит лишь о том, что информация не полна, поэтому при выводе как бы учитываются оба случая. Чтобы получить желаемый эффект, необходимо для каждого бездетного явно указать для него отсутствие ребенка с помощью утверждения *Лена*:  $\forall$  родитель\_ребенок. Nothing. Последнее означает, что мы индивидуальность *Лена* сами определили в класс *Бездетный*.

Ранее мы рассматривали определение класса дедушек, у которых внуки только мальчики:

*ДедушикаВнуковМальчиков*  $\equiv$  Мужчина

*and*(родитель\_ребенка some

(Человек

*and*(родитель\_ребенка some Мужчина)

*and*(родитель\_ребенка only Мужчина).

Если в *ABox* занести только факты, кто чей родитель и запустить резонер на выполнение, то для любой родословной мы будем получать этот класс пустым. Аналогично для классов *Мама**Только**Мальчиков*, *Папа**Только**Девочек* и т.п. Причина такого результата опять в неполноте информации для базы знаний. Необходимо указать явно, что других детей в семье нет. Это можно сделать, задав количество детей в семье следующим образом

Вика:родитель\_ребенка *exactly 1* или Володя: родитель\_ребенка *exactly 2*. Таким образом, если при заданной терминологии вывод не дает ожидаемого результата, надо проверить не только терминологию, но и проанализировать полноту информации, заданной фактами.

### 3.18. Связь с языком *OWL*

Язык веб-онтологий *OWL* [50] разрабатывается как язык, на котором можно описывать и публиковать в веб так называемые сетевые онтологии – формально записанные утверждения о понятиях и объектах некоторой предметной области. Одним из требований к таким онтологиям заключается в том, чтобы содержащиеся в них знания были «доступны» для машинной обработки, в частности, для автоматизированного логического вывода новых знаний на основе имеющихся. Для этого требуется, чтобы язык, на котором формулируются онтологии, имел точную семантику, а соответствующие логические проблемы были разрешимы (и имели практическую допустимую вычислительную сложность). Кроме того, желательно, чтобы такой язык имел довольно большую выражительную силу, пригодную для формулировки на нём практически значимых фактов.

Дескрипционные логики обладают такими свойствами, и по этой причине они были выбраны в качестве логической основы для языка веб-онтологий *OWL*. Последний является языком, имеющим *XML*-формат, поэтому можно сказать, что *OWL* является переформулировкой некоторых ДЛ с использованием синтаксиса *XML*. Поскольку существует много ДЛ, различающихся как по выразительной силе, так и по вычислительной сложности, это привело к тому, что в языке *OWL* имеется несколько вариантов.

**Соответствие терминов:** имеющиеся в ДЛ понятия *концепт*, *роль*, *индивиду* и *база знаний* в *OWL* соответствуют понятиям *класс*, *свойство*, *объект* и *онтология*, соответственно.

Официальной рекомендацией *W3C* от 10 февраля 2004 года является версия языка *OWL 1.0*. Данная спецификация языка *OWL* подразделяется на следующие варианты:

***OWL-Lite*** – соответствует дескрипционной логике *SHIF(D)*;

***OWL-DL*** – соответствует дескрипционной логике *SHOIN(D)*;

***OWL-Full*** – не соответствует какой-либо ДЛ, более того, является неразрешимым.

Версия языка ***OWL 1.1*** покрывает дескрипционную логику  $SROIQ(D)$ , включающую в себя логику  $SHOIQ(D)$ , составные аксиомы вложенности ролей в  $TBox$  (буква  $R$  в названии логики), а также аксиомы непересекаемости, рефлексивности, иррефлексивности и асимметричности ролей, универсальную роль (интерпретируемую как  $\Delta^I \times \Delta^I$ ), конструктор концепта  $\exists R.\text{Self}$  (интерпретируемый как множество элементов, являющихся  $R$ -последователем самих себя) и допускает утверждения  $\neg R(a, b)$  в  $ABox$ .

Одновременно с этим разрабатывается следующая версия языка ***OWL 2.0*** [57, 63], которая, помимо перечисленного, даст возможность формулировать онтологии в языке, соответствующем дескрипционной логике  $EL$  (преимущество которой в том, что она имеет полиномиальную вычислительную сложность); привнесет синтаксические улучшения, позволяющие легче составлять запросы к базам знаний и выдавать ответы на них; а также будет содержать механизмы для формулировки правил логического вывода.

Более подробное описание языка *OWL* рассматривается в следующей части данного пособия.

### 3.19. Машины вывода и редакторы

Имеется множество программных систем (машин вывода), позволяющих совершать логический анализ в дескрипционных логиках (проверять онтологию на непротиворечивость, строить таксономии, проверять выполнимость и вложенность концептов, делать запросы к базам знаний и др.). Подобные системы различаются по поддерживаемым ими дескрипционным логикам, по типу реализованной в них разрешающей процедуры (например, табло-алгоритм, резолюция и т.п.), по поддерживаемым форматам данных, языку программирования, на котором они реализованы, и другим параметрам. Среди наиболее известных можно перечислить системы:

*CEL* – поддерживает логику  $EL+$ , имеющую полиномиальную сложность, написана на *LISP*;

*FaCT++* – поддерживает логику  $SROIQ(D)$ , а также *OWL 2.0*, реализует табло-алгоритм, написана на *C++*;

*KAON2* – поддерживает логику *SHIQ*, расширенную специальными правилами вывода, реализует алгоритм, основанный на резолюции, написана на *Java*;

*Pellet* – поддерживает логику *SROIQ(D)*, а также *OWL 1.1*, реализует табло-алгоритм, написана на *Java*;

*RacerPro* – поддерживает логику *SROIQ(D)*, реализует табло-алгоритм, написана на *LISP*.

Создан единый ресурс – список машин ДЛ-вывода, постоянно поддерживаемый в актуальном состоянии и описывающий основные аспекты этих и других программных систем, обеспечивающих логический вывод в ДЛ.

Существуют также *редакторы* онтологий, позволяющие создавать/редактировать онтологии, сохранять их в различных форматах, некоторые позволяют подключить блок рассуждений (*reasoner*) и с его помощью произвести логический анализ онтологии. Одним из наиболее известных является редактор онтологий *Protégé*, позволяющий работать с онтологиями в языке *OWL Full*.

### 3.20. О вычислительной сложности логики *ALC*

Сведения о вычислительной сложности логических проблем в логике *ALC* приведены в табл. 3.6 [60].

В таблице ключевое слово *PSPACE* означает, что при выполнении табло-алгоритма размер необходимой памяти находится в полиномиальной зависимости от размера концепта; а ключевое слово *EXPTIME* соответствует экспоненциальной зависимости времени выполнения алгоритма от размера концепта. Под размером концепта понимается число символов, описывающих концепт.

Таблица 3.6. Вычислительная сложность логики *ALC*

Логика <i>ALC</i>	Терминология		
Проблема	пустая	ациклическая	произвольная
Выполнимость концептов	PSPACE	PSPACE	EXPTIME
Выполнимость <i>ABox</i>	PSPACE	PSPACE	EXPTIME

Из-за экспоненциальной зависимости времени выполнения алгоритма от размера концепта нет никакой гарантии, что большие (по количеству и размеру концептов) реализации закончат работу. Однако существуют системы (*FACT*, *PELLET*, *RACER*, *HERMIT*), успешно работающие на практике. Достигнуто это благодаря различным оптимизациям табло-алгоритма [14, 55].

### **Контрольные вопросы**

1. Назовите ключевые алгоритмические проблемы, связанные с конкретной ДЛ.
2. Какие свойства характеризуют конкретную ДЛ?
3. Что такое табло-алгоритм для логики ALC без терминологии?
4. Какие правила вывода используются в табло-алгоритме?
5. Как для табло-алгоритма построить дерево поиска?
6. Каковы отличия табло-алгоритма для логики ALC без терминологии и с учетом терминологии?
7. Назовите основные отличия баз знаний от баз данных.
8. Какова вычислительная сложность логики ALC?
9. Какие языки основаны на дескрипционных логиках?
10. Какие резонеры существуют на настоящий момент и каковы их особенности?

### **Контрольные задания**

1. Введя базовые концепты и роли, такие как, например, *Человек*, *Мужчина*, *Женщина*, *Дочь*, *Сын*, *Дети*, *x родитель\_ребенок u*; *x ребенок\_родителя u* и т.д., запишите на языке логики *ALC*, следующие выражения:

Дети – это сыновья или дочери;

Сын – это мужчина и ребенок родителя – человека;

Дочь – это женщина и ребенок родителя – человека;

Многодетный родитель (число детей больше или равно 2);

Человек – это мужчина или женщина;

Родитель – это человек, у которого есть дети;

Отец – это мужчина и родитель ребенка-человека;

Мать – это женщина и родитель ребенка-человека;

Бабушка – это женщина и родитель ребенка, у которого есть дети;

Дедушка – это мужчина и родитель ребенка, у которого есть дети;

Бездетный как родитель пустого множества детей;

Сестра – это дочь и ребенок многодетного родителя;

Брат – это сын и ребенок многодетного родителя;

Внук или Внучка – дети родителей, у которых тоже есть родители;

Племянник – это сын брата или сестры, т.е. хотя бы один его родитель из многодетной семьи;

Родитель только мальчиков;

Студенты, интересующиеся информатикой и не интересующиеся философией;

Студенты, не интересующиеся математикой;

Студенты, которые пьют только чай.

2. С помощью табло-алгоритма определите, реализуем ли концепт:

$(\forall \text{имеет\_ребенка.Мужчина}) \sqcap (\exists \text{имеет\_ребенка.} \neg \text{Мужчина})?$

$(\forall \text{имеет\_ребенка. Мужчина}) \sqcap (\exists \text{имеет\_ребенка.Мужчина})?$

$\forall R. (\neg C \sqcup D) \sqcap \exists R. (C \sqcap D)?$

3. Даны аксиомы:

*Женщина – это Человек и имеет\_пол Женский;*

*Мама – это Родитель и имеет\_пол Женский;*

*Родитель – это Человек и имеет\_ребенка тоже Человека*

Опишите их на языке логики ALC и с помощью табло-алгоритма докажите вложенность концептов *Mama*  $\sqsubseteq$  *Женщина*.

*Однако нелегко установить границы нашему разуму: он любознателен, жаден и столь же мало склонен остановиться, пройдя тысячу шагов, как и пройдя пятьдесят.*

*Монтень Мишель де*

## Глава 4

### **OWL – ЯЗЫК ОПИСАНИЯ ОНТОЛОГИЙ**

#### **4.1. Основные понятия**

Источниками при подготовке данного раздела о языках описания онтологий послужили следующие материалы: OWL Web Ontology Language Guide: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>, Новые возможности языка OWL 2.0: [http://shcherbak.net/translations/ru.owl2primer\\_shcherbak.net.htm](http://shcherbak.net/translations/ru.owl2primer_shcherbak.net.htm).

Язык OWL позволяет выразить информацию об окружающем мире, а затем построить определенные выводы на ее основе. Инструментальные средства OWL – блоки рассуждений (резонеры) – дают возможность автоматически сформировать эти выводы. При описании предметной области на языке OWL предполагается, что окружающий мир в основном состоит из отдельных сущностей (обычно известных как индивиды или объекты). Индивиды взаимосвязаны друг с другом и со значениями данных посредством свойств. С помощью OWL мы можем сгруппировать индивиды, обладающие определенными характеристиками, в классы.

OWL является частью Semantic Web, поэтому имена в OWL представлены в виде международных идентификаторов ресурсов (IRI, international resource identifier). Так как IRI достаточно длинный, то в OWL можно использовать компактный вариант написания, состоящий из префикса и ссылки, разделенных двоеточием. Существует множество доступных вариантов OWL-синтаксиса, служащих разнообразным целям. Когда OWL-информация передается по сети, она записана на диалекте XML.

Самым простым в восприятии (даже для неспециалиста) является Манчестерский синтаксис [40, 63]. Функциональный синтаксис [63] более прост с точки зрения спецификации и использования в рамках инструментов рассуждений. Синтаксис OWL XML – это XML-синтаксис для OWL, который определен посредством XML-

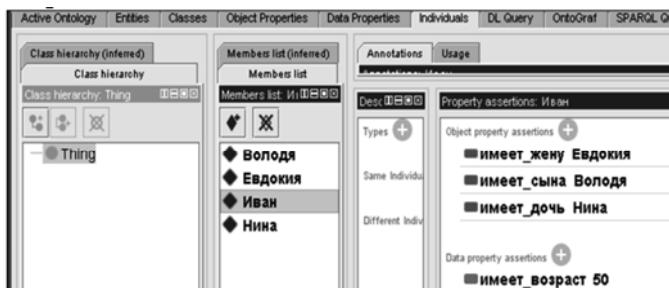
схемы [63]. Синтаксис RDF/XML для OWL представляет собой простой RDF/XML со специфическим преобразованием для OWL-конструкций [46, 48]. Есть специальные инструментальные средства перевода между разными диалектами синтаксиса OWL [67].

В дальнейшем, чтобы не загромождать основной текст пособия, конструкции языка и примеры программ рассматриваются с использованием *OWL/XML* синтаксиса. Причем сначала показывается, как представить фрагмент предметной области в редакторе Protege, а затем как его описать в программе. В приложении приведена вся программа, записанная в двух форматах, соответствующих синтаксисам: *Манчестерскому*, *Функциональному*. Там же для каждого синтаксиса даны таблицы ключевых слов с пояснением их смысла. Во всех фрагментах программ конструкция /\*Комментарий\*/ представляет собой комментарий к программе.

## 4.2. Конструкции языка OWL

С конструкциями языка познакомимся на простом примере. Положим, надо представить информацию об отдельной семье.

Сначала необходимо определить, какие индивиды фигурируют в семье, и как **они взаимосвязаны** друг с другом, какие **значения данных** связаны с ними. После этого можно записать все эти сведения при помощи *OWL*.



**Рис. 4.1.** Ввод сведений о семье Ивана – формирование АВох

1. Пусть в нашей семье родители Иван и Евдокия, их дети Нина и Володя. Все они индивидуальности в нашей онтологии (рис. 4.1). Для отображения связей между ними будем использовать роли (двуместные предикаты, описывающие бинарные отношения) «имеет\_жену», «имеет\_сына», «имеет\_дочь». А чтобы

указать возраст каждого, воспользуемся связью «имеет\_возраст» индивидуальности с данными целого типа. Всю информацию о предметной области запишем в виде фактов на языке *OWL* следующем образом:

### ***OWL XML синтаксис***

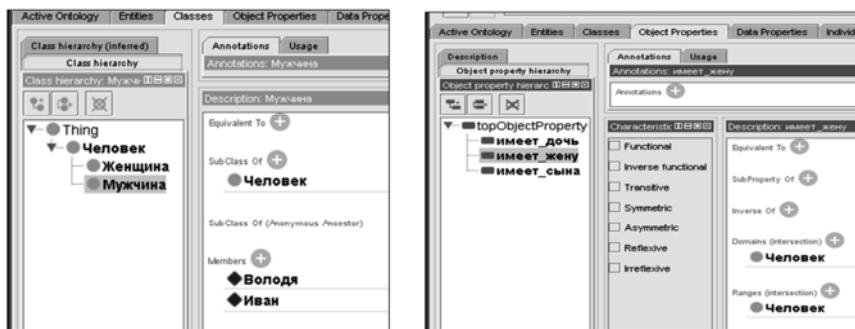
```
/* Объявление отношений «имеет_жену», «имеет_сына», «имеет_дочь»,  
«имеет_возраст» и индивидуальностей: Володя, Евдокия, Иван, Нина */  
<Declaration>  
    <ObjectProperty IRI="#имеет_дочь"/></Declaration>  
<Declaration>  
    <ObjectProperty IRI="#имеет_жену"/></Declaration>  
<Declaration>  
    <ObjectProperty IRI="#имеет_сына"/></Declaration>  
<Declaration>  
    <DataProperty IRI="#имеет_возраст"/></Declaration>  
<Declaration>  
    <NamedIndividual IRI="#Володя"/></Declaration>  
<Declaration>  
    <NamedIndividual IRI="#Евдокия"/></Declaration>  
<Declaration>  
    <NamedIndividual IRI="#Иван"/> </Declaration>  
<Declaration>  
    <NamedIndividual IRI="#Нина"/></Declaration>  
  
/*Описание связей (ObjectProperty) между индивидуальностями (Named-  
Individual) с помощью утверждений о свойствах объектов ObjectPro-  
pertyAssertion*/  
<ObjectPropertyAssertion>  
    <ObjectProperty IRI="#имеет_дочь"/>  
        <NamedIndividual IRI="#Евдокия"/>  
        <NamedIndividual IRI="#Нина"/>  
</ObjectPropertyAssertion>  
<ObjectPropertyAssertion>  
    <ObjectProperty IRI="#имеет_сына"/>  
        <NamedIndividual IRI="#Евдокия"/>  
        <NamedIndividual IRI="#Володя"/>  
</ObjectPropertyAssertion>  
<ObjectPropertyAssertion>  
    <ObjectProperty IRI="#имеет_дочь"/>  
        <NamedIndividual IRI="#Иван"/>
```

```
<NamedIndividual IRI="#Нина"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#имеет_жену"/>
        <NamedIndividual IRI="#Иван"/>
        <NamedIndividual IRI="#Евдокия"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#имеет_сына"/>
        <NamedIndividual IRI="#Иван"/>
        <NamedIndividual IRI="#Володя"/>
</ObjectPropertyAssertion>

/* Описание связей(DataProperty) индивидуальности с ее характеристиками
с помощью утверждений о свойствах данных DataPropertyAssertion*/
<DataPropertyAssertion>
    <DataProperty IRI="#имеет_возраст"/>
        <NamedIndividual IRI="#Володя"/>
        <Literal datatypeIRI="&xsd;integer">5</Literal>
</DataPropertyAssertion>
<DataPropertyAssertion>
    <DataProperty IRI="#имеет_возраст"/>
        <NamedIndividual IRI="#Евдокия"/>
        <Literal datatypeIRI="&xsd;integer">40</Literal>
</DataPropertyAssertion>
<DataPropertyAssertion>
    <DataProperty IRI="#имеет_возраст"/>
        <NamedIndividual IRI="#Иван"/>
        <Literal datatypeIRI="&xsd;integer">50</Literal>
</DataPropertyAssertion>
<DataPropertyAssertion>
    <DataProperty IRI="#имеет_возраст"/>
        <NamedIndividual IRI="#Нина"/>
        <Literal datatypeIRI="&xsd;integer">13</Literal>
</DataPropertyAssertion>
<DataPropertyRange>
    <DataProperty IRI="#имеет_возраст"/>
    <Datatype abbreviatedIRI="xsd:integer"/>
</DataPropertyRange>
</Ontology>
```

2. В приведенном фрагменте с помощью фактов отражены сведения о конкретной семье. Для их описания использовалась в основном *RDF* составляющая *OWL*: с помощью тегов задавались индивиды и их связь с другими индивидами посредством **свойств** (*ObjectProperty* и *DataProperty*). *OWL* также позволяет описать общие правила формирования семей. Этот процесс называется представлением знаний.

Каковы же эти правила? Заметим, что все индивиды семьи – это люди, каждый из них либо мужчина, либо женщина. В результате можно выделить общий класс Человек, а в нем классы Мужчина и Женщина. Ниже представлена информация о некоторых свойствах (отношениях). «Иметь\_жену» (рис. 4.2) это отношение между людьми (Человеком и Человеком), то есть и доменом (Domain) и диапазоном (Range) для «иметь\_жену» будет Человек, как собственно для «иметь\_сына» и «иметь\_дочь». «Иметь\_возраст» это связь Человека с характеристикой, задаваемой типом данных *integer*. Каждый индивид относится либо к классу Мужчина, либо к классу Женщина (см. рис. 4.2)



**Рис. 4.2.** Создание классов и описание доменов и диапазонов ролей  
*OWL XML синтаксис*

В программу, описанную на предыдущем шаге, добавляются следующие строки:

```
/* Объявление классов */  
<Declaration>  
    <Class IRI="#Женщина"/>  
</Declaration>  
<Declaration>
```

```
<Class IRI="#Мужчина"/>
</Declaration>
<Declaration>
    <Class IRI="#Человек"/>
</Declaration>
/* Описание отношения подкласс класса */  

<SubClassOf>
    <Class IRI="#Женщина"/>
    <Class IRI="#Человек"/>
</SubClassOf>
<SubClassOf>
    <Class IRI="#Мужчина"/>
    <Class IRI="#Человек"/>
</SubClassOf>
/* Описание принадлежности индивидуальности к классу */  

<ClassAssertion>
    <Class IRI="#Мужчина"/>
    <NamedIndividual IRI="#Володя"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Женщина"/>
    <NamedIndividual IRI="#Евдокия"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Мужчина"/>
    <NamedIndividual IRI="#Иван"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Женщина"/>
    <NamedIndividual IRI="#Нина"/>
</ClassAssertion>
/* Описание для каждого отношения (ObjectProperty) области определения (домена) и области значений (range) */  

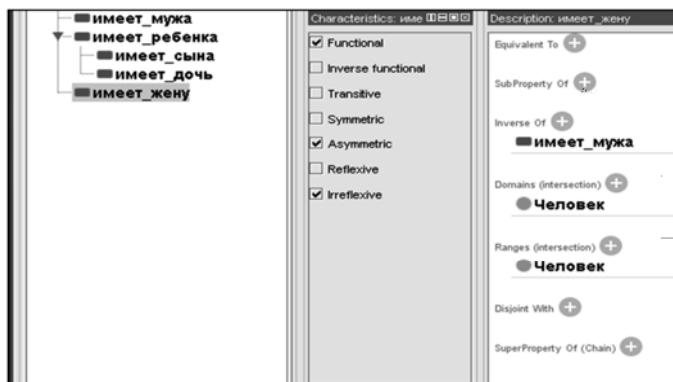
<ObjectPropertyDomain>
    <ObjectProperty IRI="#имеет_дочь"/>
    <Class IRI="#Человек"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
    <ObjectProperty IRI="#имеет_жену"/>
    <Class IRI="#Человек"/>
</ObjectPropertyDomain>
<ObjectPropertyDomain>
```

```
<ObjectProperty IRI="#имеет_сына"/>
    <Class IRI="#Человек"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
    <ObjectProperty IRI="#имеет_дочь"/>
        <Class IRI="#Человек"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#имеет_жену"/>
        <Class IRI="#Человек"/>
</ObjectPropertyRange>
<ObjectPropertyRange>
    <ObjectProperty IRI="#имеет_сына"/>
        <Class IRI="#Человек"/>
</ObjectPropertyRange>
/* Описание для каждого свойства данных (DataProperty) области
определения (домена) и области значений (range) */
<DataPropertyDomain>
    <DataProperty IRI="#имеет_возраст"/>
        <Class IRI="#Человек"/>
</DataPropertyDomain>
<DataPropertyRange>
    <DataProperty IRI="#имеет_возраст"/>
        <Datatype abbreviatedIRI="xsd:integer"/>
</DataPropertyRange>
```

Из вышеприведенной информации механизм рассуждений (резонер) может вывести, что Иван и Володя принадлежат к классу Человек, так как Иван и Володя – мужчины, а класс Мужчина является подклассом класса Человек. Аналогично Евдокия и Нина будут отнесены к классу Человек, так как они женщины, а класс Женщины – это подкласс класса Человек. То, что Иван и Володя – мужчины, а Евдокия и Нина – женщины в программе явно указано с помощью тега <ClassAssertion>.

3. Даже из этого небольшого описания семейных отношений можно сделать несколько дополнительных выводов. Например, обратным свойству «имеет\_жену» является свойство «имеет\_мужа». Так же, как свойства «имеет\_сына» и «имеет\_дочь» будут подчиненными для отношения «имеет\_ребенка». Далее, ни один индивид не может быть одновременно и мужчиной, и женщиной, то есть эти классы непересекающиеся. В большинстве случаев

индивидуам присуще единственное значение возраста, поэтому отношение «имеет\_возраст» обладает функциональным свойством данных (т.е. является функцией). По большей части индивидам присуще только одно отношение «имеет\_жену», и ни один индивид не может быть женой самому себе. Поэтому свойство «имеет\_жену» функциональное, обратно функциональное, и нерефлексивное (рис. 4.3). Также отношение «имеет\_жену» является асимметричным. Таким образом, мы расширили информацию о некоторых свойствах.



**Рис. 4.3.** Задание основных характеристик свойств

Наша программа дополнится следующими строками.

### OWL XML синтаксис

```
/* Непересекающиеся классы DisjointClasses */  

<DisjointClasses>  

    <Class IRI="#Женщина"/>  

    <Class IRI="#Мужчина"/>  

</DisjointClasses>  

/* Описание отношения подсвойство (SubObjectPropertyOf) подчинено  

   свойству */  

<SubObjectPropertyOf>  

    <ObjectProperty IRI="#имеет_дочь"/>  

    <ObjectProperty IRI="#имеет_ребенка"/>  

</SubObjectPropertyOf>  

<SubObjectPropertyOf>  

    <ObjectProperty IRI="#имеет_сына"/>  

    <ObjectProperty IRI="#имеет_ребенка"/>
```

```
</SubObjectPropertyOf>
/* Описание обратного (инверсного) свойства */  

<InverseObjectProperties>
    <ObjectProperty IRI="#имеет_мужа"/>
    <ObjectProperty IRI="#имеет_жену"/>
</InverseObjectProperties>
/* Описание функционального свойства */  

<FunctionalObjectProperty>
    <ObjectProperty IRI="#имеет_жену"/>
</FunctionalObjectProperty>
/* Описание обратного свойства как функционального */  

<InverseFunctionalObjectProperty>
    <ObjectProperty IRI="#имеет_мужа"/>
</InverseFunctionalObjectProperty>
/* Описание асимметричного свойства */  

<AsymmetricObjectProperty>
    <ObjectProperty IRI="#имеет_жену"/>
</AsymmetricObjectProperty>
<AsymmetricObjectProperty>
    <ObjectProperty IRI="#имеет_мужа"/>
</AsymmetricObjectProperty>
/* Описание нерефлексивного свойства */  

<IrreflexiveObjectProperty>
    <ObjectProperty IRI="#имеет_жену"/>
</IrreflexiveObjectProperty>
<IrreflexiveObjectProperty>
    <ObjectProperty IRI="#имеет_мужа"/>
</IrreflexiveObjectProperty>
/* Описание для отношения ("имеет_ребенка") области определения
(домена) и области значений (range) */  

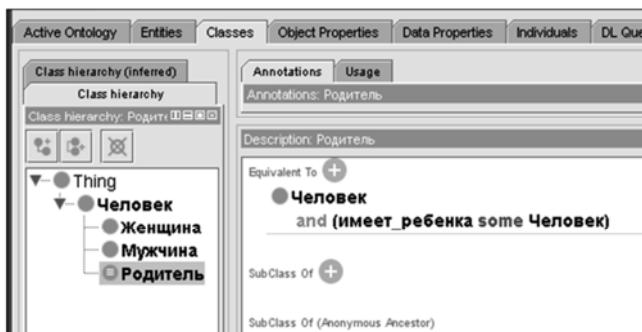
<ObjectPropertyDomain>
    <ObjectProperty IRI="#имеет_ребенка"/>
    <Class IRI="#Человек"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
    <ObjectProperty IRI="#имеет_ребенка"/>
    <Class IRI="#Человек"/>
</ObjectPropertyRange>
/* Непрекращающиеся свойства */  

<DisjointObjectProperties>
    <ObjectProperty IRI="#имеет_дочь"/>
    <ObjectProperty IRI="#имеет_сына"/>
```

```
</DisjointObjectProperties>
```

Вышеприведенная информация о семьях вообще и о конкретной семье в частности имеет целый ряд следствий. К примеру, если свойство «имеет\_мужа» является обратным свойству «имеет\_жену», то мужем для Евдокии будет Иван. В этом можно убедиться, выполнив вывод с помощью резонера и открыв вкладку *Individuals*. Инструментальные средства рассуждений *OWL* позволяют установить возможность вывода определенных следствий из имеющейся информации.

4. Таким образом, мы записали довольно много информации о семьях, используя только три класса: *Женщина*, *Мужчина* и *Человек*. *OWL* – это мощный язык описания классов с использованием аксиом. Введем класс *Родитель* и опишем его с помощью аксиомы *Родитель*  $\equiv$  Человек *and* имеет\_ребенка *some* Человек (рис. 4.4).



**Рис. 4.4.** Описание класса *Родитель*  
с помощью аксиомы эквивалентности

В программе это будет выглядеть следующим образом.

### OWL XML синтаксис

```
<Declaration>
    <Class IRI="#Родитель"/>
</Declaration>
/* Аксиома эквивалентности */
```

	*/
--	----

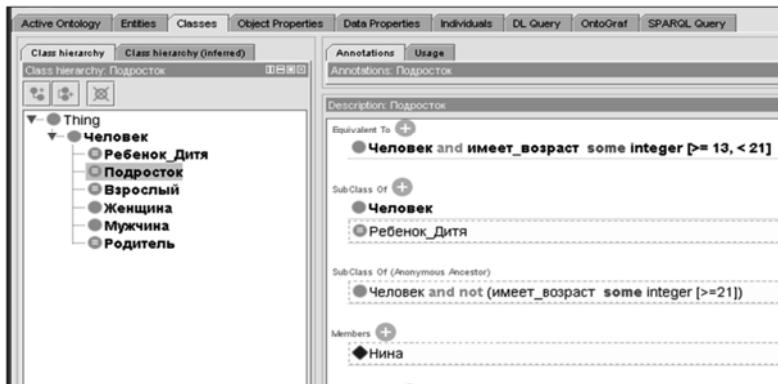
```
<EquivalentClasses>
    <Class IRI="#Родитель"/>
    <ObjectIntersectionOf> /* Пересечение (and) */
        <Class IRI="#Человек"/>
        <ObjectSomeValuesFrom> /*Существует (some) */
```

```

<ObjectProperty IRI="#имеет_ребенка"/>
    <Class IRI="#Человек"/>
</ObjectSomeValuesFrom>
</ObjectIntersectionOf>
</EquivalentClasses>

```

Выполнив вывод с помощью резонера, получим, что к родителям относятся Иван и Евдокия, что *Родитель* является подклассом класса *Человек*.



**Рис. 4.5.** Описание класса *Подросток* с помощью аксиомы эквивалентности

5. Еще *OWL* позволяет представить информацию о некоторых группах значений данных, называемых *диапазонами данных*. Так, к примеру (рис. 4.5), *Подросток* (*Юный*) может обозначать тех людей, чей возраст является целым числом, большим 13, но не превышающим 20. К классу *Взрослый* отнесем тех людей, у которых значение возраста равно хотя бы 21, а к группе *Ребенок* (*Дитя*) причислим тех, у кого возраст является дополнением к взрослому возрасту.

В программе описание вновь введенных классов

*Подросток*  $\equiv$  Человек *and* имеет\_возраст some integer [ $\geq 13, < 21$ ];  
*Взрослый*  $\equiv$  Человек *and* имеет\_возраст some integer [ $\geq 21$ ];  
*Ребенок\_Дитя*  $\equiv$  Человек *and*

*not*(имеет\_возраст some integer [ $\geq 21$ ])

будет выглядеть следующим образом.

### **OWL XML синтаксис**

```

<Declaration>
    <Class IRI="#Взрослый"/>
</Declaration>
<Declaration>
    <Class IRI="#Подросток"/>
</Declaration>
<Declaration>
    <Class IRI="#Ребенок_Дитя"/>
</Declaration>
/* Аксиомы эквивалентности */ *
<EquivalentClasses>
    <Class IRI="#Взрослый"/>
    <ObjectIntersectionOf>
        <Class IRI="#Человек"/>
        <DataSomeValuesFrom>
            <DataProperty IRI="#имеет_возраст"/>
                /*Описание диапазона значений >=21*/
                <DatatypeRestriction>
                    <Datatype abbreviatedIRI="xsd:integer"/>
                    <FacetRestriction facet="&xsd:minInclusive">
                        <Literal datatypeIRI="&xsd;integer">21</Literal>
                    </FacetRestriction>
                </DatatypeRestriction>
            </DataSomeValuesFrom>
        </ObjectIntersectionOf>
    </EquivalentClasses>
    <EquivalentClasses>
        <Class IRI="#Подросток"/>
        <ObjectIntersectionOf>
            <Class IRI="#Человек"/>
            <DataSomeValuesFrom>
                <DataProperty IRI="#имеет_возраст"/>
                    /*Описание диапазона значений (>=13) и (<21)*/
                    <DatatypeRestriction>
                        <Datatype abbreviatedIRI="xsd:integer"/>
                        <FacetRestriction facet="&xsd:maxExclusive">
                            <Literal datatypeIRI="&xsd;integer">21</Literal>
                        </FacetRestriction>
                        <FacetRestriction facet="&xsd:minInclusive">
                            <Literal datatypeIRI="&xsd;integer">13</Literal>
                        </FacetRestriction>
                    </DatatypeRestriction>
            </DataSomeValuesFrom>
        </ObjectIntersectionOf>
    </EquivalentClasses>

```

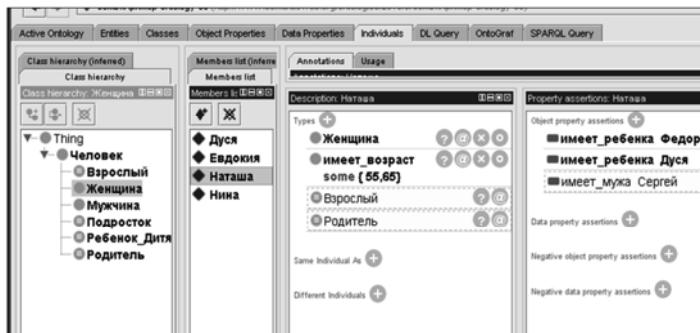
```
</FacetRestriction>
    </DatatypeRestriction>
</DataSomeValuesFrom>
</ObjectIntersectionOf>
</EquivalentClasses>
<EquivalentClasses>
    <Class IRI="#Ребенок_Дитя"/>
    <ObjectIntersectionOf>
        <Class IRI="#Человек"/>
        <ObjectComplementOf>
            <DataSomeValuesFrom>
                <DataProperty IRI="#имеет_возраст"/>
                    <DatatypeRestriction>
                        <Datatype abbreviatedIRI="xsd:integer"/>
                        <FacetRestriction facet="'.minInclusive'>
                            <Literal datatypeIRI="xsd:integer">21</Literal>
                        </FacetRestriction>
                    </DatatypeRestriction>
                </DataSomeValuesFrom>
                <ObjectComplementOf>
                    <ObjectIntersectionOf>
</EquivalentClasses>
<SubClassOf>
    <Class IRI="#Взрослый"/>
    <Class IRI="#Человек"/>
</SubClassOf>
```

Резонер выведет, что Нина относится к группе *Подросток*, но не к Взрослым. Оба Иван и Евдокия – *Взрослые*, но не *Подросток*. Володя не относится ни к Взрослым, ни к *Подросткам*. Нина и Володя являются детьми (*Ребенок\_Дитя*).

Для обозначения диапазонов данных в *OWL* применяются встроенные типы данных из *XML*-схемы, например, *xsd:integer*. Еще несколько полезных типов данных – *xsd:string* и *xsd:decimal*.

6. До настоящего времени рассматривалось применение *OWL* с точки зрения языка структурирования данных. Однако на самом деле *OWL* гораздо более выразителен, нежели просто языки структурирования данных. *OWL* обладает рядом преимуществ. Некоторые из них обуславливают разницу между *OWL* и другими формализмами. Вот почему так важно понимание отличий *OWL* от других технологий.

В предыдущем примере была известна вся информация о возрасте членов семьи. Но *OWL* предназначен для работы в условиях неполной (или неточной) информации. Рассмотрим еще одну семью. Отец Сергей, жена Наташа и их дети Федор и Дуся. Для членов этой семьи задана неполная информация о возрасте. Известно, что возраст Сергея 77 лет, Наташе либо 60 , либо 55 лет, Дусе не 37 лет, а Федору от 15 до 20 лет. В редакторе эта информация будет задаваться для каждой индивидуальности с помощью типа данных, описываемого логической формулой (рис. 4.6), а именно: для Наташи – *имеет\_возраст some {55, 65}*; для Дуси – *not (имеет\_возраст value 37)*; для Федора – *имеет\_возраст some integer[>= 15, < 21]*.



**Рис. 4.6.** Приближенное описание возраста с помощью типа данных

Сведения о новой семье и их возрасте в программе будут описаны следующими строками.

### OWL XML синтаксис

```
<Declaration>                                <NamedIndividual IRI="#Дуся"/>
</Declaration>
<Declaration>                                <NamedIndividual IRI="#Наташа"/>
</Declaration>
<Declaration>                                <NamedIndividual IRI="#Сергей"/>
</Declaration>
<Declaration>                                <NamedIndividual IRI="#Федор"/>
</Declaration>
```

```
/* Описание принадлежности элемента классу*/
<ClassAssertion>
    <Class IRI="#Женщина"/>
        <NamedIndividual IRI="#Дуся"/>
</ClassAssertion>
<ClassAssertion>
    <ObjectComplementOf> /* Отрицание (дополнение)*/
        <DataHasValue> /* Имеет значение */
            <DataProperty IRI="#имеет_возраст"/>
/* Определение конкретного(37) значения данного */
    <Literal datatypeIRI="xsd:integer">37</Literal>
        </DataHasValue>
    </ObjectComplementOf>
        <NamedIndividual IRI="#Дуся"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Женщина"/>
        <NamedIndividual IRI="#Наташа"/>
</ClassAssertion>
<ClassAssertion>
    <DataSomeValuesFrom>
        <DataProperty IRI="#имеет_возраст"/>
            <DataOneOf> /* Одно из списка значений */
                <Literal datatypeIRI="xsd:integer">55</Literal>
                <Literal datatypeIRI="xsd:integer">65</Literal>
            </DataOneOf>
        </DataSomeValuesFrom>
        <NamedIndividual IRI="#Наташа"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Мужчина"/>
        <NamedIndividual IRI="#Сергей"/>
</ClassAssertion>
<ClassAssertion>
    <Class IRI="#Мужчина"/>
        <NamedIndividual IRI="#Федор"/>
</ClassAssertion>
<ClassAssertion>
    <DataSomeValuesFrom>
        <DataProperty IRI="#имеет_возраст"/>
            <DatatypeRestriction>
                <Datatype abbreviatedIRI="xsd:integer"/>
```

```
/* Диапазон >= 15 */
<FacetRestriction facet="&xsd:minInclusive">
<Literal datatypeIRI="&xsd;integer">15</Literal>
</FacetRestriction>
/* Диапазон <=21 */
<FacetRestriction facet="&xsd:maxExclusive">
<Literal datatypeIRI="&xsd;integer">21</Literal>
</FacetRestriction>
</DatatypeRestriction>
</DataSomeValuesFrom>
<NamedIndividual IRI="#Федор"/>
</ClassAssertion>
<ObjectPropertyAssertion>
<ObjectProperty IRI="#имеет_ребенка">
<NamedIndividual IRI="#Наташа"/>
<NamedIndividual IRI="#Федор"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
<ObjectProperty IRI="#имеет_ребенка">
<NamedIndividual IRI="#Наташа"/>
<NamedIndividual IRI="#Дуся"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
<ObjectProperty IRI="#имеет_жену">
<NamedIndividual IRI="#Сергей"/>
<NamedIndividual IRI="#Наташа"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
<ObjectProperty IRI="#имеет_ребенка">
<NamedIndividual IRI="#Сергей"/>
<NamedIndividual IRI="#Федор"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
<ObjectProperty IRI="#имеет_ребенка">
<NamedIndividual IRI="#Сергей"/>
<NamedIndividual IRI="#Дуся"/>
</ObjectPropertyAssertion>
<DataPropertyAssertion>
<DataProperty IRI="#имеет_возраст">
<NamedIndividual IRI="#Сергей"/>
<Literal datatypeIRI="&xsd;integer">77</Literal>
</DataPropertyAssertion>
```

В результате вывода с помощью резонера даже при неполной информации получим, что Сергей и Наташа относятся к классу *Взрослый*, Федор к классу *Подросток*, а вот к какому классу принадлежит Дуся остается неопределенным.

7. Существует множество причин неполноты в *OWL*. Некоторые из них могут показаться неожиданными. Например, несмотря на то, что наличие двух детей у Сергея кажется очевидным, это вовсе не так. Также не очевидно, что у Сергея есть, по меньшей мере, один ребенок, который принадлежит к классу *Мужчина*.

Ведь ничего не сказано о том, что Федор и Дуся – единственные дети Сергея. В *OWL* не предпринимается никаких допущений касательно того, что если что-то не сказано, то это неправда. Можно констатировать, что у Сергея больше нет детей. Это можно сделать несколькими способами. Первый из них состоит в явном указании, что у Сергея ровно 2 ребенка.

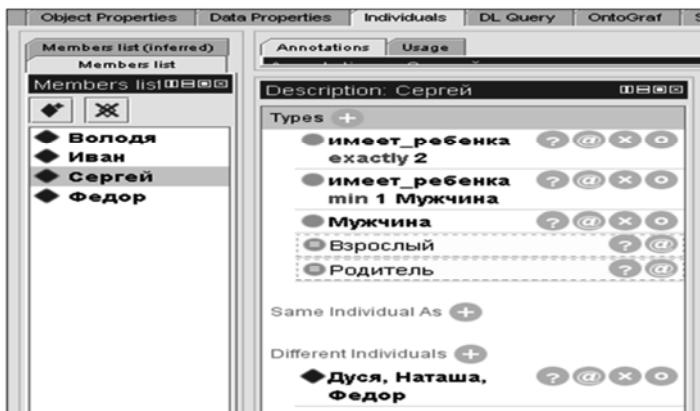


Рис. 4.7. Более точное описание индивидуальностей

Однако, даже наличие у Сергея одного ребенка, принадлежащего к классу *Мужчина*, не приводит к адекватному выводу. В онтологии нет информации, что Федор и Дуся – разные люди, и ничего из сказанного ранее не указывает на то, что они не одно лицо. *OWL* не делает никаких допущений, что разные имена принадлежат разным индивидам. (Это «допущение уникальных имен» особенно опасно в *Semantic Web*, где имена могут быть даны разными организациями в разное время, бессознательно ссылаясь на один и тот же индивид). Если Федор и Дуся – одно лицо, тогда у

Сергея должен быть другой ребенок, принадлежащий к классу *Мужчина*.

Можно возразить, что Федор и Дуся – разные люди, ведь они принадлежат к разным классам по половому признаку. К сожалению, это не так, если не установить, что *Женщина* и *Мужчина* различны, т.е. эти множества не пересекаются.

Однако добавлять информацию надо таким образом, чтобы она не была избыточной. Например, вовсе не обязательно добавлять информацию о различии членов семьи Ивана. Наличие разных возрастов предполагает, что все это – разные люди. Аналогично жены и их мужья – разные люди, потому как свойство «иметь\_жену» является нерефлексивным.

Также можно установить, что два имени ссылаются на (обозначают) один и тот же индивид. Например, можно сказать, что Дуся и Евдокия – это один индивид.

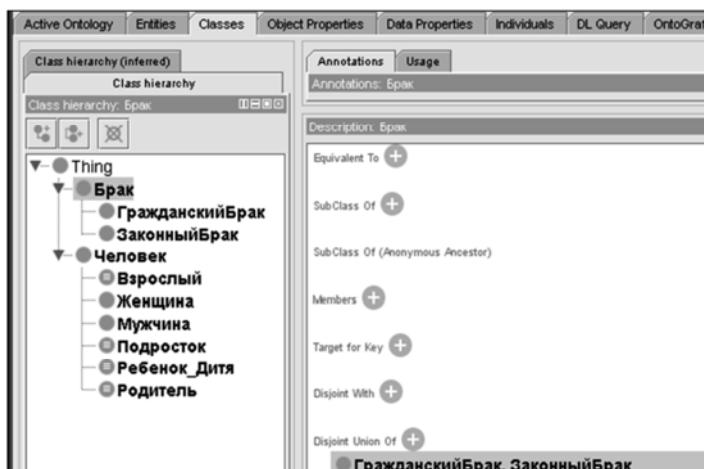
Таким образом, в следующем фрагменте добавим информацию, что у Сергея два ребенка и как минимум у него один мальчик (рис. 4.7), что Федор и Дуся различные индивидуальности, а Дуся и Евдокия – одно и то же лицо. Объявим классы *Мужчина* и *Женщина* – непересекающимися.

### **OWL XML синтаксис**

```
<DisjointClasses>
    <Class IRI="#Женщина"/>
    <Class IRI="#Мужчина"/>
</DisjointClasses>
<ClassAssertion>
/* Число связей указанного тунна >= 1(ObjectMinCardinality) */
    <ObjectMinCardinality cardinality="1">
        <ObjectProperty IRI="#имеет_ребенка"/>
        <Class IRI="#Мужчина"/>
    </ObjectMinCardinality>
    <NamedIndividual IRI="#Сергей"/>
</ClassAssertion>
<ClassAssertion>
    <ObjectExactCardinality cardinality="2">
        <ObjectProperty IRI="#имеет_ребенка"/>
    </ObjectExactCardinality>
    <NamedIndividual IRI="#Сергей"/>
</ClassAssertion>
```

```
<SameIndividual>
    <NamedIndividual IRI="#Дуся"/>
    <NamedIndividual IRI="#Евдокия"/>
</SameIndividual>
<DifferentIndividuals>
    <NamedIndividual IRI="#Дуся"/>
    <NamedIndividual IRI="#Наташа"/>
    <NamedIndividual IRI="#Сергей"/>
    <NamedIndividual IRI="#Федор"/>
</DifferentIndividuals>
```

8. В программе показано, что *Мужчина* и *Женщина* – непересекающиеся множества, то есть не существует индивидов, принадлежащих к обоим множествам. Зачастую, это необходимо указывать для примитивных классов.



**Рис. 4.8.** Метод *DisjointUnion* для описания объединения непересекающихся классов

Так, например, для законного брака (ЗаконныйБрак) и гражданского брака (ГражданскийБрак) нужно задать их непересекаемость. Можно установить, что Брак является объединением их обоих. Исходя из того, что подобная ситуация встречается достаточно часто, когда класс является объединением нескольких непересекающихся классов, для краткости записи в OWL предусмотрен специальный метод *DisjointUnion* (рис. 4.8).

### OWL XML синтаксис

```
<DisjointUnion>
    <Class IRI="#Брак"/>
        <Class IRI="#ГражданскийБрак"/>
        <Class IRI="#ЗаконныйБрак"/>
</DisjointUnion>
```

9. В OWL могут присутствовать транзитивные свойства (*transitive properties*), то есть свойства (как, например, наличие предка – «имеет\_предка»), которые представляют собой обобщение обратного свойства (в данном случае – наличие ребенка – «имеет\_ребенка»), и являются нерефлексивными (рис. 4.9).

Существуют и другие разновидности информации, которую можно добавить о свойствах. Можем создать свойство «иметь\_супруга» (супруг(и)), как симметричное (*symmetric*) и нерефлексивное обобщение свойства «иметь\_жену».

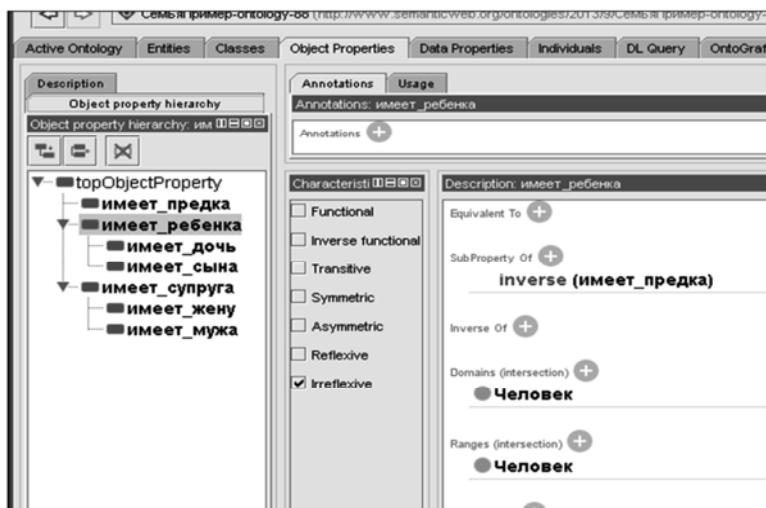


Рис. 4.9. Описание отношений «имеет\_предка» и «имеет\_супруга»

Можно сделать вывод, что «иметь\_супруга» также является обобщением для «иметь\_мужа». Ведь «иметь\_супруга» – симметричное обобщение обратного свойства к свойству «иметь\_мужа».

**OWL XML синтаксис**

```

<Declaration>
    <Class IRI="#Потомок"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#имеет_предка"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#имеет_супруга"/>
</Declaration>
<EquivalentClasses>
    <Class IRI="#Потомок"/>
    <ObjectIntersectionOf>
        <Class IRI="#Человек"/>
        <ObjectHasValue>
            <ObjectProperty IRI="#имеет_предка"/>
            <NamedIndividual IRI="#Наташа"/>
            </ObjectHasValue>
        </ObjectIntersectionOf>
    </EquivalentClasses>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="#имеет_жену"/>
        <ObjectProperty IRI="#имеет_супруга"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="#имеет_мужа"/>
        <ObjectProperty IRI="#имеет_супруга"/>
    </SubObjectPropertyOf>
    <SubObjectPropertyOf>
        <ObjectProperty IRI="#имеет_ребенка"/>
        <ObjectInverseOf>
            <ObjectProperty IRI="#имеет_предка"/>
        </ObjectInverseOf>
    </SubObjectPropertyOf>
    <SymmetricObjectProperty>
        <ObjectProperty IRI="#имеет_супруга"/>
    </SymmetricObjectProperty>
/* Описание транзитивного свойства */ *
<TransitiveObjectProperty>
    <ObjectProperty IRI="#имеет_предка"/>
</TransitiveObjectProperty>
```

```
<IrreflexiveObjectProperty>
    <ObjectProperty IRI="#имеет_супруга"/>
</IrreflexiveObjectProperty>
```

Из вышеприведенной информации можно получить вывод, что Сергей является предком для Володи, причем Сергей не его непосредственный родитель. В программе определен класс

*Потомок ≡ Человек and (имеет\_предка value Наташа).*

Он описан так, что позволяет получить потомков Наташи – это Володя, Дуся, Евдокия, Федор и Нина.

10. Добавим свойство «любит». Заметим, что оно несимметричное. Опишем новый класс: *Нарциссит ≡ Человек and любит some Self*, обозначающий самовлюбленных людей (рис. 4.10). Добавим для некоторых индивидуальностей немного информации о том, кого или что они любят. Например: Иван любит Евдокию, а Володя любит Володю, т.е. самого себя (рис. 4.10). В результате работы машины вывода получим, что Володя принадлежит к классу *Нарциссит*.



Рис. 4.10. Описание класса *Нарциссит*.  
Индивидуальность Володя любит самого себя

В программе эта информация приобретет вид:

### OWL XML синтаксис

```
<Declaration>
    <Class IRI="#Нарциссит"/>
</Declaration>
<EquivalentClasses>
    <Class IRI="#Нарциссит"/>
        <ObjectIntersectionOf>
            <Class IRI="#Человек"/>
```

```

<ObjectHasSelf> /* Связь с самим собой */
<ObjectProperty IRI="#любит"/>
    </ObjectHasSelf>
</ObjectIntersectionOf>

</EquivalentClasses>
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#любит"/>
        <NamedIndividual IRI="#Володя"/>
        <NamedIndividual IRI="#Володя"/>
</ObjectPropertyAssertion>
<ObjectPropertyAssertion>
    <ObjectProperty IRI="#любит"/>
        <NamedIndividual IRI="#Иван"/>
        <NamedIndividual IRI="#Евдокия"/>
</ObjectPropertyAssertion>
<AsymmetricObjectProperty>
    <ObjectProperty IRI="#любит"/>
</AsymmetricObjectProperty>

```

11. Еще несколько слов можно сказать об объединении свойств в *OWL* посредством цепочек объектных свойств. Например, можем создать в виде цепочки свойств (рис. 4.11) отношение *имеет\_ребенка o имеет\_ребенка* ⊑ *имеет\_внуков*. Затем определить класс *ДедИлиБабушка* ≡ *Человек and имеет\_внуков some Человек*.



**Рис. 4.11.** Определение свойства «имеет\_внуков» с помощью цепочки свойств

### OWL XML синтаксис

```
<Declaration>                                <Class IRI="#ДедИлиБабушка"/>
</Declaration>
<Declaration>                                <ObjectProperty IRI="#имеет_внуков"/>
</Declaration>
<EquivalentClasses>
  <Class IRI="#ДедИлиБабушка"/>
  <ObjectIntersectionOf>
    <Class IRI="#Человек"/>
    <ObjectSomeValuesFrom>
      <ObjectProperty IRI="#имеет_внуков"/>
        <Class IRI="#Человек"/>
      </ObjectSomeValuesFrom>
    </ObjectIntersectionOf>
  </EquivalentClasses>
  <SubObjectPropertyOf>
    <ObjectPropertyChain> /*Цепочка(композиция) свойств*/
      <ObjectProperty IRI="#имеет_ребенка"/>
      <ObjectProperty IRI="#имеет_ребенка"/>
    </ObjectPropertyChain>
    <ObjectProperty IRI="#имеет_внуков"/>
  </SubObjectPropertyOf>
```

### 4.3. Управление онтологиями

Информация, которой мы оперировали до настоящего момента, подпадает под две категории. Первая – это общая информация о классах и свойствах, касающихся семейных отношений, а вторая – это частная информация по двум взаимосвязанным семьям. Общая информация в *OWL* объединяется в онтологию (*ontology*), которая впоследствии используется различными приложениями. Можно задавать имена для *OWL*-онтологий, которые в основном совпадают с местоположением документа онтологии в глобальной сети. Частная информация также может быть помещена в онтологию, если эта информация используется разными приложениями.

```
<Ontology ...ontologyIRI=
  "http://www.semanticweb.org/ontologies/2013/9/Ce
   мьяПример-ontology-88">.....
</Ontology>
```

*OWL*-онтологии размещают внутрь *OWL*-документов, которые затем располагают в локальной файловой системе или выкладывают в сеть. Кроме *OWL*-онтологий, документы *OWL* содержат информацию о преобразовании коротких имен, используемых в *OWL*-онтологиях (например, IRI="#Женичина"), в идентификаторы IRI, предоставляя пространство для префиксов. Поэтому IRI – это объединение пространства для префиксов и ссылки. В нашем примере онтологии описание префиксов приведено в следующем фрагменте программы.

### ***OWL XML синтаксис***

```
<!DOCTYPE Ontology [  
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">  
  <!ENTITY xml "http://www.w3.org/XML/1998/namespace">  
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">  
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">  
>  
<Ontology xmlns="http://www.w3.org/2002/07/owl#"  
  xml:base="http://www.semanticweb.org/ontologies/2013/9/СемьяПример-  
  ontology-88"  
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"  
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"  
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"  
  xmlns:xml="http://www.w3.org/XML/1998/namespace"  
  ontologyIRI="http://www.semanticweb.org/ontologies/2013/9/СемьяПример-  
  ontology-88"> ...  
<Prefix name=""  
  IRI="http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-  
  88#">  
<Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#">  
<Prefix name="rdf" IRI="http://www.w3.org/1999/02/22-rdf-syntax-ns#">  
<Prefix name="xsd" IRI="http://www.w3.org/2001/XMLSchema#">  
<Prefix name="rdfs" IRI="http://www.w3.org/2000/01/rdf-schema#">  
<Prefix name="СемьяПример-ontology-88"  
  IRI="http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-  
  88#">  
.....  
</Ontology>
```

В *OWL* также принято повторно использовать общую информацию в других онтологиях. Вместо требования копировать эту информацию, *OWL* позволяет импортировать содержимое

целых онтологий в другие онтологии посредством объявлений импорта, как в примере ниже:

### **OWL XML синтаксис**

```
<Import  
IRI="http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-  
88" />  
<Import>  
http://www.semanticweb.org/ontologies/2013/5/semia  
</Import>
```

Так как *Semantic Web* и конструкции онтологий являются распределенными, для онтологий считается общепринятым использование разных имен для одинаковых концепций, свойств или индивидов. Некоторые конструкции *OWL* могут применяться для определения того, что разные имена ссылаются на один и тот же класс, свойство или индивид.

Так, например, можно выполнить привязку имен, задействованных в нашей онтологии, к именам импортированной онтологии «<http://www.semanticweb.org/ontologies/2013/5/semia>», как в примере ниже:

### **OWL XML синтаксис**

```
<EquivalentClasses>  
    <Class  
IRI="http://www.semanticweb.org/ontologies/2013/5/semia#Женщины"/>  
        <Class IRI="#Женщина"/>  
</EquivalentClasses>  
<EquivalentClasses>  
    <Class  
IRI="http://www.semanticweb.org/ontologies/2013/5/semia#Мужчины"/>  
        <Class IRI="#Мужчина"/>  
</EquivalentClasses>  
<EquivalentObjectProperties>  
    <ObjectProperty  
IRI="http://www.semanticweb.org/ontologies/2013/5/semia#дедушка_или_ба-  
бушка_ребенка"/>  
        <ObjectProperty IRI="#имеет_внуков"/>  
</EquivalentObjectProperties>  
<EquivalentObjectProperties>  
    <ObjectProperty
```

```

IRI="http://www.semanticweb.org/ontologies/2013/5/semia#ребенок_родителя"/>
    <ObjectProperty IRI="#имеет_ребенка"/>
</EquivalentObjectProperties>
<SameIndividual>
    <NamedIndividual
IRI="http://www.semanticweb.org/ontologies/2013/5/semia#Евдокия"/>
    <NamedIndividual IRI="#Евдокия"/>
</SameIndividual>
<SameIndividual>
    <NamedIndividual
IRI="http://www.semanticweb.org/ontologies/2013/5/semia#Нина"/>
    <NamedIndividual IRI="#Нина"/>
</SameIndividual>

```

Во многих случаях необходимо пояснить отдельные части *OWL*-онтологии. Для этих целей в *OWL* предназначены аннотации (annotations). *OWL*-аннотация попросту связывает пары свойство-значение с частями либо с целой онтологией. На самом деле эта информация не является частью логического содержания онтологии. В следующем фрагменте программы приведены примеры аннотаций для всей онтологии, для индивидуальности Дуся (рис. 4.12), для класса *Нарциссум*.

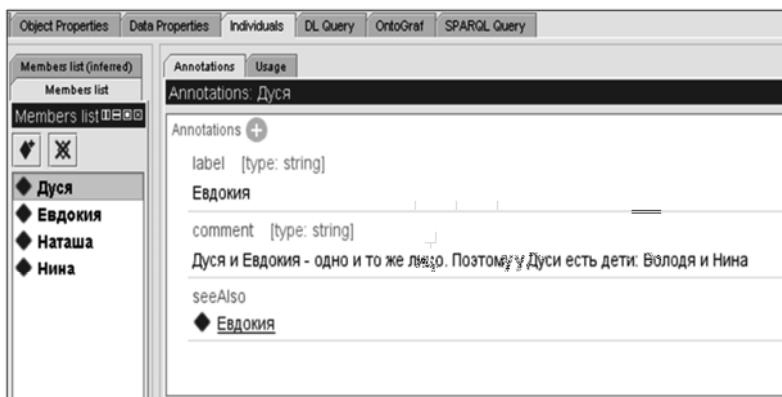


Рис. 4.12. Аннотация к индивидуальности Дуся

```

/* Аннотация ко всей онтологии поясняет ее назначение */
<Annotation>
    <AnnotationProperty abbreviatedIRI="rdfs:comment"/>
    <Literal datatypeIRI="&xsd:string">

```

Онтология семьи. В ней общие знания о семье применяются к двум семьям, для которых заданы индивидуальности и их отношения друг к другу.

```
</Literal>
</Annotation>
/* Предыдущая версия, совместимая с текущей онтологией */ 
<Annotation>
<AnnotationProperty
    abbreviatedIRI="owl:backwardCompatibleWith">
    <Literal datatypeIRI="&rdf;PlainLiteral">
http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-88
</Literal>
</Annotation>
/* Аннотация к индивидуальности Дуся */ 
/* Аннотация – комментарий */ 
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:comment">
        <IRI>#Дуся</IRI>
        <Literal datatypeIRI="&xsd:string">
```

Дуся и Евдокия – одно и то же лицо. Поэтому у Дуси есть дети: Володя и Нина

```
</Literal>
</AnnotationAssertion>
/* Аннотация – метка задает альтернативное имя */ 
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:label">
        <IRI>#Дуся</IRI>
        <Literal datatypeIRI="&xsd:string">Евдокия</Literal>
</AnnotationAssertion>
/* Аннотация – ссылка «смотри также» */ 
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:seeAlso">
        <IRI>#Дуся</IRI>
        <IRI>#Евдокия</IRI>
</AnnotationAssertion>
/* Аннотация к классу Нарциссит */ 
/* Аннотация – комментарий */ 
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:comment">
        <IRI>#Нарциссит</IRI>
```

```

<Literal datatypeIRI="&rdf;PlainLiteral">
    Человек, который любит самого себя
</Literal>
/* Аннотация – метка задает альтернативное имя */ 
</AnnotationAssertion>
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:label"/>
    <IRI>#Нарциссит</IRI>
    <Literal datatypeIRI="&rdf;PlainLiteral">
        Самовлюбленный
    </Literal>
</AnnotationAssertion>
/* Аннотация – ссылка «смотри также» */ 
<AnnotationAssertion>
    <AnnotationProperty abbreviatedIRI="rdfs:seeAlso"/>
    <IRI>#Нарциссит</IRI>
    <IRI>#Володя</IRI>
</AnnotationAssertion>

```

Для поддержки управления онтологиями в *OWL* есть понятие деклараций. Основная идея состоит в том, что каждый класс, свойство или индивид должны быть задекларированы в онтологии, после чего они могут использоваться в данной онтологии или других, в которые импортирована эта.

В манчестерском синтаксисе декларации являются неявными. Конструкции, представляющие информацию о классе, свойстве или индивиде неявно декларируют эти элементы при необходимости. Другие разновидности синтаксиса предполагают явные декларации. Ниже приведены примеры деклараций.

### **OWL XML синтаксис**

```

<Declaration>
    <Class IRI="#Потомок"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#имеет_предка"/>
</Declaration>
<Declaration>
    <ObjectProperty IRI="#имеет_супруга"/>
</Declaration>

```

В *OWL* есть еще несколько интересных элементов, которые, однако, не вошли в данный пример. Более подробная информация по этим конструкциям находится в документе Структурная спецификация и функциональный синтаксис *OWL 2* [63].

Вся программа приведена в приложении в двух форматах, соответствующих двум синтаксисам.

Написание и отладка такой программы является трудоемким и рутинным процессом. Помощником в этом случае может быть редактор *Protege 4*. С его помощью удобно разрабатывать и проверять онтологию, а затем ее можно сохранить в любом необходимом формате. Так, например, редактор *Protege 4.2* позволяет сохранить онтологию в восьми форматах.

#### **4.4. Связь OWL с другими технологиями**

##### **Технологии описания ресурсов RDF и RDFS**

Среди всех технологий *RDF(S)* ближе всего стоит к *OWL*. Обе они основываются на логическом представлении знаний. В большинстве случаев *RDF(S)* может рассматриваться как подмножество *OWL*. И, конечно, *RDF/XML* является исходным синтаксисом обмена для *OWL*.

##### **XML**

Технологии *OWL* и *XML* разделяют несколько общих моментов [69]. Во-первых, *OWL* может быть выражен с помощью *XML* (например, *RDF/XML* или *XML*-синтаксис для *OWL*), а, следовательно, и обработан посредством инструментальных средств для *XML*. Во-вторых, *OWL* позволяет повторно использовать типы данных и специальные ограничения (*facet*) для определения производных типов данных *XML*-схемы. Также в *OWL* существует несколько вариантов определения типов *XML*-схемы. И, наконец, обе технологии (*OWL* и *XML*) успешно применяются для концептуального моделирования и описания данных. Хотя используемые при этом методы несколько отличаются в рамках каждой из технологий. По сравнению с *XML* язык *OWL* ориентирован на более абстрактное и высокоуровневое концептуальное моделирование.

*OWL* предназначен для поддержки выявления связей между классами посредством автоматизированных рассуждений. В языке *Web*-онтологий заложено гораздо меньше допущений касательно

описываемых сущностей – как в целом, так и в терминах их физической реализации в вычислительных системах.

Обе технологии обеспечивают поддержку высокого уровня абстракции. Однако *XML*-схема больше нацелена на организацию данных. Ведь главная задача этой технологии заключается в валидации *XML*-документов.

### **Базы данных**

Онтологии на базе *OWL* гораздо мощнее и гибче, нежели схемы баз данных. В основном схемы баз данных лишь определяют те виды информации, которые могут быть связаны с объектами (или кортежами), принадлежащими какому-либо классу (или таблице). В *OWL*-онтологиях классы могут предназначаться для аналогичных целей, однако они также могут содержать условия по распознаванию, поэтому явная типизация в *OWL* необязательна. Безусловно, наличие подобной гибкости в *OWL* при определении типов может потребовать сложных выводов.

Последнее важное различие между базами данных и *OWL* заключается в том, что информация, хранимая в базе, определяется посредством схемы базы данных и ограничений целостности. Если схема не поддерживает хранение определенных типов информации или информация нарушает принятые ограничения целостности, тогда эта информация не может быть сохранена в базе данных. С другой стороны, *OWL* позволяет связать информацию произвольного типа практически с любым объектом, если в онтологии ничего не противоречит этой связи. Поэтому с точки зрения хранения информации *OWL* гораздо гибче. Последнее важное различие между базами данных и *OWL* заключается в том, что информация, хранимая в базе, определяется посредством схемы базы данных и ограничений целостности. Если схема не поддерживает хранение определенных типов информации или информация нарушает принятые ограничения целостности, тогда эта информация не может быть сохранена в базе данных. С другой стороны, *OWL* позволяет связать информацию произвольного типа практически с любым объектом, если в онтологии ничего не противоречит этой связи. Поэтому с точки зрения хранения информации *OWL* гораздо гибче.

## Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) также характеризуется моделированием относительно объекта, и поэтому имеет много общего с *OWL*. Однако ООП в основном используется в контексте полной информации. Здесь информация об объекте, которая потенциально может быть получена, ограничена типом этого объекта. Как и с базами данных, основное отличие ООП от *OWL* состоит в разной степени полноты информации об объекте. Аналогично классы в ООП гораздо менее экспрессивны, чем *OWL*-классы.

Более того, язык *Web*-онтологий является строго декларативным и логическим. Следовательно, *OWL* не присущи компоненты ООП, как, например, методы. Рассуждения в *OWL* основаны на четкой логике, где нет ничего схожего на наследование, а уж тем более на наследование с исключениями или с переопределением.

## OWL 1

*OWL 2* представляет собой доработанный язык *Web*-онтологий (*OWL1*), совместимый с предыдущей версией. В *OWL 2* добавлено несколько новых конструкций для расширения экспрессивности языка, в том числе конструкции по ограничению кардинальности, цепочки ролей и предикаты эксплицитных данных. *OWL 2* также включает новую *XML*-сериализацию (нацелен на взаимодействие с набором инструментов *XML*, то есть *XSLT*, языками схем и т. д.) и набор подмножеств профилей с разнообразным применением и вычислительными свойствами.

Для тех, кто интересуется *OWL1*, стоит ознакомиться с соответствующими документами по *OWL1*: Обзор и Руководство по языку [67]. Исходя из того, что каждая онтология *OWL 1* является также онтологией *OWL 2*, в документации по *OWL 1* [67] содержится небольшое введение в *OWL 2* (хотя только по синтаксису *RDF/XML*). Существует и другая документация по *OWL 1*, которая доступна на сайте рабочей группы *WebOnt* [70].

## Контрольные вопросы

1. Что позволяет выразить язык *OWL*, и на каком формализме он основан?
2. Что такое *URI*, *IRI*?

3. Какие форматы используются для описания онтологий на языке *OWL*?
4. Какие теги используются для описания объектных свойств и свойств данных?
5. Какие теги используются для определения индивидуальностей?
6. Как задать для конкретной индивидуальности ее свойства?
7. Какие теги используются для определения классов, подклассов?
8. Как индивидуальность приписать к какому-либо классу?
9. Как описать характеристики свойств: функциональность, транзитивность, симметричность, рефлексивность и т.п.?
10. Как описываются аксиомы включения и аксиомы эквивалентности?

### **Контрольные задания**

1. На языке *OWL/XML* описать свою семью, определить классы *Мать*, *Отец*, *Дочь*, *Сын*.
2. Для разработанной онтологии получить ее описание в заданном формате (*Манчестерский синтаксис*, *Функциональный синтаксис*, *Синтаксис OWL/XML*, *Синтаксис RDF/XML*) и объяснить назначение конструкций программы.

*Любое решение проблемы порождает новую проблему.*

*Иоганн Вольфганг Гете*

## Г л а в а 5

### ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ ОНТОЛОГИЙ

В настоящее время онтологии, как специализированные базы знаний, находят применение во многих областях науки и техники. Процесс разработки онтологии в наибольшей степени зависит от области ее применения, от того, какой цели будет служить разрабатываемая онтология.

Онтологии могут быть использованы везде, где требуется обработка данных, учитывая их семантику. Построение онтологии часто не является само по себе конечной целью, обычно онтологии далее используются другими программами для решения практических целей. На данном этапе развития науки существует ряд задач, где применение онтологий может дать хорошие результаты.

В следующих подразделах описываются области применения онтологий, в которых они оказываются наиболее полезными.

#### 5.1. Основные области применения онтологий

1. Системы искусственного интеллекта [7, 14, 20, 27]:
  - a) создание и использование баз знаний (БЗ);
  - b) создание систем, реализующих механизмы рассуждений (ЭС, системы управления, интеллектуальные роботы) [38, 45];
  - c) создание точных и по возможности непротиворечивых определений значений каждого термина на основе логики первого порядка;
  - d) определение семантики с помощью множества аксиом, которые автоматически позволяют получать ответ на множество вопросов о предметной области.
2. Информационный поиск [1, 12, 64]:
  - a) организация эффективного поиска в базах данных, информационных каталогах, базах знаний [1, 24, 25];
  - b) организация поиска по смыслу в текстовой информации.

денция к максимальному отображению лексики естественного языка [13, 23];

с) семантический анализ текста (извлечение знаний из текста). Поддержка лингвистически ориентированных информационных технологий: автоматический перевод, системы распознавания (OCR и Speech Recognition); корректоры текста; автоматизация аннотирования и реферирования; диалог на естественном языке (ЕЯ);

д) организация управляемого данными регламентированного диалога [64];

- е) машинный перевод;
- ф) вопросно-ответные системы.

3. Semantic Web [3, 23, 64]:

а) представление смысла в метаданных об Информационном Ресурсе;

б) если существующая Web-сеть – это огромное множество документов, которые связаны перекрестными ссылками, то создаваемая Семантическая Сеть должна добавить к существующей сети множество онтологий и метаописаний знаний, содержащихся в документах Web-сети;

- с) семантический поиск в Internet;

д) описание содержимого Web – страниц. Онтологии содержимого Web-страниц необходимы поисковым программам для улучшения качества поиска по Web. Одним из главных компонентов будущего Web будет Semantic Web, в котором каждая Web-страница предоставляет также онтологию своего содержимого [5, 17].

4. Разработка терминологии [12, 42]:

а) обеспечение общей терминологии для множества специалистов и совместно используемых приложений. Онтология рассматривается как терминосистема [42];

б) для совместного использования людьми или программными агентами общего понимания структуры информации.

5. Системы управления знаниями [5, 6, 8, 10, 14, 18, 22, 28, 56]:

а) построение и использование баз общих знаний для различных информационных систем (ИС);

б) многократное применение баз знаний и информационных массивов, представляющих сведения о технических системах на различных стадиях их жизненного цикла;

- с) объединение двух приложений баз данных с разными схемами, но с близкими концептуальными моделями;
- д) объединение несовместимых информационных систем;
- е) онтология для хранилищ данных;
- ф) использование для организации и поиска знаний о предметной области;
- г) интеграция знаний в системах извлечения знаний.

6. Базы данных [7, 8, 64]:

- а) расширение запросов;
- б) поддержка доступа по специальным ограничениям;
- в) упорядочивание записей;
- г) направление пользователя по определенному пути.

7. Концептуальное моделирование [4, 7, 64]:

а) концептуальный доступ к информационным ресурсам: модель интерфейса приложения; семантический поиск документов, концептуальные схемы БД, вопросно-ответные системы и др. Сюда же можно отнести задачу интеграции разнородных ресурсов.

8. Разработка классификаторов [58]:

а) Классификаторы являются нормативными документами, распределяющими технико-экономическую информацию в соответствии с ее классификацией (классами, группами, видами и другими группировками). Классификаторы являются обязательными для применения при создании государственных информационных систем и информационных ресурсов и межведомственном обмене информацией, а также в правовых актах в социально-экономической области для однозначной идентификации объектов правоотношений. Социально-экономическая область охватывает, в том числе прогнозирование, статистический учет, банковскую деятельность.

9. Онтологии в образовании [5, 9, 12]:

а) применение web-онтологий в задачах дистанционного обучения [15, 16];  
б) онтологический подход к менеджменту ресурсов вуза [19];  
в) применение онтологий при построении тестов;  
г) управление качеством образовательного процесса на основе онтологии [25];  
е) базовые онтологии портала знаний [55];  
ф) разработка онтологии учебно-методических комплексов [24, 25].

### 10. Мультиагентные системы [59]:

а) разработка коммуникационных моделей взаимодействия агентов-учредителей системы и заинтересованных агентов внешней среды на домене сотрудничества, основанная на XML-сообщениях и программном посреднике, позволяющая рассматривать Internet как распределённую вычислительную платформу мультиагентной системы и использовать коммуникационные Internet-технологии, обеспечивающие надёжную передачу данных [59];

б) формальное отображение семантики домена предметной области на структуру XML-сообщений.

### 11. Системы поддержки принятия решений [53]:

а) медицинские системы [56] поддержки принятия решений.

Рассмотрим некоторые из вышеприведенных направлений более подробно.

## 5.2. Системы искусственного интеллекта



**Рис. 5.1.** Обобщенная схема интеллектуальной системы (редактора онтологий)

На рис. 5.1 приведена обобщенная схема интеллектуальной системы. Ядром большинства подобных систем являются два блока: база знаний (БЗ) и машина вывода. Они тесно взаимосвязаны между собой. Алгоритм работы машины вывода зависит от выбранной в базе знаний модели представления знаний.

Пользователь обращается к системе с вопросом. Ответ на него получает машина вывода, обращаясь к базе знаний, где находятся наиболее общие знания о предметной области вопроса.

Онтология – это одна из моделей представления знаний. Эта модель объединила в себе черты логических моделей и наглядность семантических сетей. Формальной моделью представления знаний в БЗ является дескрипционная логика (DL), концепты и аксиомы которой записываются на языке OWL.

Можно сказать, что редакторы онтологий являются интеллектуальными системами, построенными по принципу, представленному на рис. 5.1. Редакторы онтологий – это системы программных элементов (плагинов), каждый из которых отвечает за свою функцию и в любой момент может быть обновлен, добавлен или удален. К наиболее важным плагинам можно отнести сам графический редактор, машину вывода (резонер) и *ОнтоГраф* – построитель семантических сетей. С помощью графического редактора концепты, их свойства и аксиомы, описывающие предметную область, заносятся в базу знаний на понятном для пользователя языке. Машина вывода получает ответ на запрос пользователя, устанавливает вложенность одного класса в другой, распределяет индивидуальности по классам предметной области. В настоящее время существует несколько вариантов резонеров (*Fact++*, *HermiT* и т.п.), и каждый из них непрерывно совершенствуется в направлении оптимизации процедуры вывода. В основе алгоритма работы одних лежит принцип резолюции, в основе других – табло-алгоритм. *ОнтоГраф* отвечает за представление знаний о предметной области в виде семантической сети. База знаний (онтология) может быть сохранена в виде файла на языке *OWL* в различных форматах и передана по сети любому заинтересованному потребителю, любой другой системе независимо от архитектуры ее программных и аппаратных средств.

Онтологии служат своеобразной моделью окружающего мира, а их структура такова, что утверждения о предметной области легко поддаются машинной обработке и анализу. Онтологии снабжают систему сведениями о хорошо описанной семантике заданных слов и указывают иерархическое строение области, взаимосвязь элементов. Все это позволяет компьютерным программам при помощи онтологии делать умозаключения из представленной информации и манипулировать ими. Онтологии могут использоваться для вывода умозаключений, необходимых для понимания текстов на глубинно-семантическом уровне, что требуется для высококачественного машинного перевода и может служить базой для расширения и уточнения информационного поиска.

### 5.3. Semantic Web

Идея Семантической Сети (*Semantic Web*) впервые была провозглашена в 2001 году Тимом Бернерсом-Ли (создателем *World Wide Web*) [3, 47, 49, 69]. Однако она не является новой ни для автора, ни для *web*-сообщества в целом. Суть ее состоит в автоматизации «интеллектуальных» задач обработки значения (в семантическом смысле) тех или иных ресурсов, имеющихся в Сети. Обработкой и обменом информации должны заниматься не люди, а специальные интеллектуальные агенты (программы, размещенные в Сети). Но для того, чтобы взаимодействовать между собой, агенты должны иметь общее (разделяемое всеми) формальное представление значения для любого ресурса. Именно для цели представления общей, явной и формальной спецификации значения в *Semantic Web* используются онтологии.

Формальная спецификация содержимого *Web*-документа дает возможность поисковой программе делать выводы о соответствии поискового запроса данному *Web*-документу не только на основе синтаксической информации, получаемой из текста этого документа, но и основываясь на семантике содержания данного документа. Это может кардинально улучшить качество *Web*-поиска, так как описание мира *Web*-страницы, понятное поисковой программе, дает последней гораздо больше информации, чем она может получить из неструктурированного текста.

За годы, прошедшие с момента первой публикации о *Semantic Web*, был разработан целый ряд стандартов и рекомендаций, реализовано множество проектов. Но, несмотря на отдельные успехи, до сих пор (и это признает сам Т. Бернерс-Ли) нельзя сказать, что идея *Semantic Web* реализована на практике.

Работа над средствами описания семантики в Сети началась задолго до публикации 2001 года. В 1997 году консорциум *W3C* определил спецификацию *RDF* (*Resource Description Framework*). *RDF* предоставляет простой, но мощный язык описания ресурсов, основанный на триплетах (*triple-based*) «Субъект-Предикат-Объект» и спецификации *URI*. В 1999 году *RDF* получает статус рекомендации. Этот шаг в направлении улучшения функциональности и обеспечения интероперабельности (т.е. возможности обмениваться данными несмотря на их разнородность) в Сети считается одним из

важнейших. Концептуально *RDF* дает минимальный уровень для представления знаний в Сети. Спецификация *RDF* опирается на ранние стандарты, лежащие в основе *Web*:

- *Unicode* служит для представления символов алфавитов различных языков;
- *URI* используется для определения уникальных идентификаторов ресурсов;
- *XML* и *XML Schema* – для структурирования и обмена информацией и для хранения *RDF* (*XML* синтаксис *RDF*).

Кроме *RDF* был разработан язык описания структурированных словарей для *RDF – RDF Schema* (*RDFS*). Он предоставляет минимальный набор средств для спецификации онтологий. *RDFS* получил статус рекомендации *W3C* в 2004 году. Однако препятствием для *Semantic Web* стало то, что документов, написанных на языке *RDF/RDFS*, было относительно мало. В период с 2001 по 2004 годы шла интенсивная работа по созданию программных средств для обработки и автоматической генерации *RDF*-документов.

Результатом в 2004 году стал язык *GRDDL* (*Gleaning Resource Descriptions from Dialects of Languages*). Его назначение состоит в предоставлении средств для извлечения *RDF*-триплетов из *XML* и *XHTML* данных (в особенности это относится к документам, автоматически генерируемым из закрытых баз данных). Развивалось и программное обеспечение для *Semantic Web*. В области создания библиотек классов и построения логических выводов над *RDF*-графами была создана библиотека *Jena Framework*, в области создания модулей расширения для браузеров – *Smile* для *Firefox*. В области создания визуальных сред редактирования большое число редакторов онтологий стали поддерживать *RDF*.

В 2004 году статус рекомендации получил язык *OWL* (*Web Ontology Language*). Он имеет 3 диалекта (3 множества структурных единиц), используемых в зависимости от требуемой выразительной мощности. *OWL* фактически является надстройкой над *RDF/RDFS* и поддерживает эффективное представление онтологий в терминах классов и свойств, обеспечение простых логических проверок целостности онтологий и связывание онтологий друг с другом (импорт внешних определений). Многие формализмы описания знаний могут быть отображены на формализм *OWL* (два из его диалектов – *OWL Lite* и *OWL DL* – соответствуют двум деск-

риптивным логикам, имеющим разную выразительную силу). Большое число создаваемых в настоящее время онтологий кодируются на *OWL*; уже существующие онтологии транслируются в него.

На этом работа по обеспечению *Semantic Web* необходимыми стандартами не остановилась. В 2005 году началась работа над форматом обмена правилами – *RIF* (*Rule Interchange Format*). Его назначение – соединить в одном стандарте несколько формализмов для описания правил (по которым может осуществляться нетривиальный логический вывод): логику клауз Хорна, логики высших порядков, продукционные модели и т.п.

Язык *SPARQL* – язык запросов к *RDF*-хранилищам – в январе 2008 года приобрел статус официальной рекомендации Консорциума *W3C*. Синтаксически он очень похож на *SQL*. Он уже широко используется разработчиками информационных систем.

На рис. 5.2 представлена диаграмма, называемая иногда стеком (или даже «слоеным пирогом») *Semantic Web*.

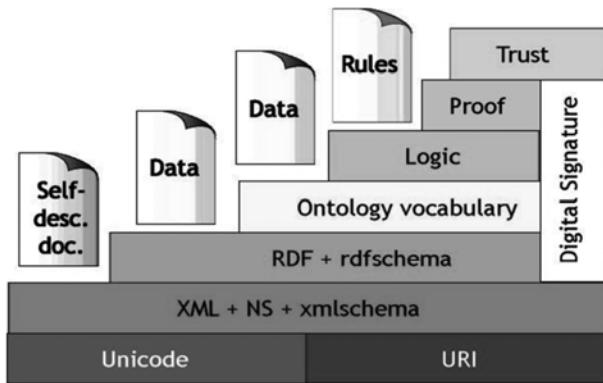


Рис. 5.2. Диаграмма *Semantic Web*

Все основные уровни диаграммы были описаны выше. Уровнями «*Ontology vocabulary*» и «*Logic*» соответствуют *OWL* и *RIF*. Уровень «*Trust*» на данный момент остается незатронутым никакими стандартами. Здесь и возникает одно из существенных препятствий к реализации всей идеи: поддержка автоматической проверки корректности и правдивости информации. В самом деле, у многих поставщиков семантических описаний может возникнуть соблазн «обмануть» программу-агента, предоставив информацию,

не соответствующую действительности, либо навязчивую рекламу, как это в настоящее время проделывается с поисковыми машинами, спам-фильтрами и т.п.

Еще одним камнем преткновения для создания *Semantic Web* является фактическое отсутствие работающих интеллектуальных агентов. Не всякая программа, обрабатывающая *RDF*, является агентом *Semantic Web*, точно так же как и не всякая программа, написанная на ПРОЛОГЕ, является приложением в области искусственного интеллекта.

Семантическая Сеть продолжает развиваться – появляются новые стандарты.

Новый шаг – начало разработки формата обмена правилами *RIF*, построенными над онтологиями, и определение требований и области его применения. Появилось множество свободно распространяемых библиотек для разработки приложений «под Semantic Web». Главными задачами, стоящими перед сообществом Семантической Сети, остаются создание новых онтологий и согласование существующих.

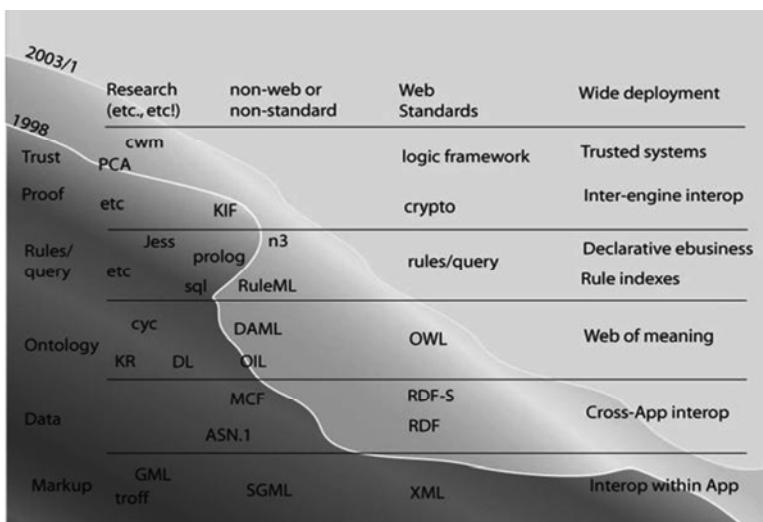


Рис. 5.3. «Приливная волна» *Semantic Web*

На рис. 5.3 наглядно видна тенденция последних 3-7 лет, которую можно условно назвать «прилив Семантической Сети» [47].

## 5.4. Разработка и управление терминологией

Терминология, управляемые словари, таксономия и тому подобные вещи используются для целого ряда задач информационного поиска (*IR, information retrieval*), например, расширение запросов или поддержка доступа по специальным ограничениям (*facet*). Предопределенная терминология также может применяться для упорядочивания записей (при поддержке *IR*) или направления пользователя по определенному пути. К примеру, в медицинских системах поддержки принятия решений определенная форма или ее часть отображается на экране, когда врач собирается выполнить специфические процедуры, которые в свою очередь определяются путем выявления комбинации терминов из управляющего словаря.

### Онтология – терминосистема

В современном мире обмен информацией и документацией возрастает в геометрической прогрессии. Важной составляющей научно-технического и экономического прогресса при этом становится терминология, которая играет решающую роль в профессиональной коммуникации. Растущий объем информации приводит к постоянному увеличению количества терминов. Рост числа терминов различных наук обгоняет рост числа общеупотребительных слов, поэтому уже в настоящее время число терминов отдельных наук превышает число неспециальных слов языка. Так, например, в начале 20 века вся научно-техническая терминология в немецком языке насчитывала около 3,5 млн. терминов. На сегодняшний день только в области электротехники в немецком языке более 4 млн. терминов. В то же время специальная лексика русского, английского, немецкого и других развитых языков в большинстве областей знания не представляет собой упорядоченной системы, которая соответствовала бы современному уровню науки и отвечала бы требованиям практики. Широко распространены такие негативные явления, как синонимия терминов, различное толкование терминов представителями разных научных школ и направлений, отсутствие четких определений, вариативность форм одних и тех же терминов, обилие заимствованных иноязычных терминов.

Упорядочение специальной лексики имеет огромное значение для взаимопонимания специалистов, подготовки научных и технических кадров, издания научной и справочной литературы, разви-

тия международных экономических и научных связей, получения и обмена информацией. Данная проблематика становится еще более актуальной в условиях двуязычной или многоязычной коммуникации и приводит к необходимости обширной семантизации и систематизации терминологической лексики и создания двуязычных или многоязычных глоссариев или словарей. Единственно возможный путь упорядочения терминологической лексики – это построение онтологий, терминосистем и создание терминологических банков данных.

В последнее время эксперты, занимающиеся терминологией предметных отраслей, все чаще обращаются к разработке онтологий. Онтология определяет общий словарь ученых, которые совместно используют информацию своей отрасли знания. Онтологии позволяют не только представить структуру информации, но и отделить теоретические знания от оперативных, а также анализировать знания предметной области.

## 5.5. Концептуальное моделирование

Основное преимущество *OWL* состоит в поддержке моделирования предметной области, то есть в *концептуальном моделировании*. *OWL* является мощным языком концептуального моделирования. С его помощью, к примеру, можно легко закодировать большую часть схем сущность-связь и *UML*-диаграмм. Закодировав информацию единожды, блоки рассуждений *OWL* могут находить скрытые отношения, конфликты и пропущенные элементы. Так как *OWL* позволяет описывать неполную информацию и взаимодействовать с ней, он отлично подходит для высокоуровневого концептуального моделирования, при котором происходит абстрагирование на физическом или логическом уровне информационной системы. *OWL* позволяет отложить моделирование некоторых решений, в то время как можно продолжить эффективно использовать то, что уже хорошо известно.

*OWL* поддерживает разные стили моделирования, например, «сверху вниз» или «снизу вверх», итеративное или прямолинейное, ориентированное на уточнение или, наоборот, на обобщение. В процессе моделирования механизм рассуждений (и другие инструменты) обеспечивает непрерывное взаимодействие с моделью. В действительности, порой отсутствие какой-либо реакции механизма рассуждений

дает очень ценную информацию субъекту, осуществляющему моделирование (то есть модель описана хуже, чем ожидалось).

Интерфейс большинства приложений основан на иерархии различных меню или рабочих окон. Правильно построить эту иерархию могут помочь концептуальные модели на базе *OWL*. Они могут быть положены в основу интерфейса приложений, работающих с базами данных. При этом для каждого класса пользователей может быть разработана своя онтология, соответствующая информационным потребностям пользователя, а на ее базе и свой интерфейс.

Концептуальные модели на базе *OWL* могут использоваться для объединения информации. Положим, к примеру, возникла необходимость объединения двух приложений баз данных с совершенно разными схемами, однако сходными (во всяком случае, на первый взгляд) концептуальными моделями. При трансформации обеих моделей в *OWL* и соединении их друг с другом могут быть найдены как скрытые отношения, так и противоречивость моделей. Исходя из того, что модели имеют четкую семантику, они могут систематически проверяться. Чтобы не заниматься утомительной проверкой соответствий вручную, можно задать необходимые настройки моделирования.

Концептуальные модели на основе *OWL* используются непосредственно для объединения несовместимых информационных систем. Для этого существует несколько методов: начиная с рассмотрения концептуальных моделей в виде высокоуровневых схем для хранилищ данных на основе *RDF*, и заканчивая использованием концептуальных моделей для построения распределенных запросов в домашних системах данных.

Когда *OWL*-информация передается по сети, она записана на диалекте *XML*.

## **5.6. Системы управления знаниями**

Онтологии нашли наиболее широкое применение в области систем управления знаниями [15, 18, 19, 22].

### **Организационные порталы знаний**

Несмотря на то, что разработано уже много онтологий, отражающих знания о самых разнообразных объектах, при описании

конкретных субъектов экономической деятельности надо учитывать их специфику и вносить ее в соответствующие онтологические модели.



**Рис. 5.4.** Онтологическое представление знаний о субъектах экономической деятельности

Онтологическое представление знаний о субъектах экономической деятельности, которые входят в состав какой-либо системы, можно использовать для объединения их информационных ресурсов в единое информационное пространство (рис. 5.4).

Онтология предприятия содержит классы понятий с заданными на них семантическими отношениями. Она состоит из набора технологических онтологий и организационной онтологии, отражающей организационно-функциональную структуру предприятия: состав штатного расписания (работники, администрация, обслуживающий персонал), партнеры, ресурсы и т. п. и отношения между ними. Онтологии технологий содержат понятия, описывающие производственные процессы. Общие знания предметной области, к которой относятся субъекты экономической деятельности, отображает онтология отрасли.

Разработанные онтологии позволяют сотрудникам одной отрасли или корпорации использовать общую терминологию и избежать взаимных недоразумений, которые могут усложнить сотрудничество и привести к серьезным убыткам (например, организационная онтология четко отражает взаимную иерархию и связи между подразделениями предприятия, а также сферы их компетенции, а ссылки на определенные нормативные документы обеспечивают одинаковую базу для переговоров). Они обеспечат работу со структурированными источниками данных, для которых может быть построена схема данных, то есть, описаны типы данных и связи между ними, и существует формальный способ получения отдельных элементов данных. Примерами структурированных источников данных можно считать различные базы данных (например, реляционные и объектные), а также слабо структурированные ресурсы, описанные в форматах *XML*, *RDF*, *OWL*, *DAML+OIL*.

Примером практического использования онтологических моделей технологий является система *ONTOLOGIC*, рассмотренная в разделе «Примеры существующих проектов и систем».

## 5.7. Интеграция разнородных источников данных

Под базой данных (БД) будем понимать коллекцию согласованных взаимосвязанных данных, которые имеют некоторое «скрытое (внутри) значение». БД похожи на базы знаний (БЗ), поскольку они также используются для описания некоторой предметной области с целью хранения, обработки и доступа к необходимой информации о ней. Однако есть и различия. Базы данных содержат (и способны обрабатывать) большие массивы относительно простой информации (при этом доступ возможен только к этим явно введенным данным). В базах знаний [64] обычно хранится меньший объем информации, но они имеют более сложную структуру, что позволяет использовать возможности логического вывода и получать такие утверждения, которые не были в явном виде введены.

Подразумевается, что к любой БЗ могут быть применимы 3 операции: определить (*define*), сказать (*tell*) (т.е. сделать утверждение) и спросить (*ask*). Каждая из операций может использовать один или более собственных языков, например язык описания схем и ограничений, язык обновления (для новых утверждений), язык

запросов и язык описания результата. Преимущества применения дескриптивной логики (*DL*) для улучшения каждого из языков широко изучены в литературе по базам знаний. Рассмотрим три важные задачи, возникающие при управлении данными:

- разработка концептуальной модели предметной области (онтологии) для конкретного источника данных;
- интеграция нескольких источников;
- описание и выполнение запросов.

Для каждой из задач существует подход с использованием *DL*.

При разработке БД возникает необходимость описать предметную область таким языком, чтобы описание было понятно как обычным пользователям, так и разработчикам. Это описание выполняется на языке высокого уровня и имеет форму требований. В области БД таким языком являются *ER*-диаграммы (*ER*-модель). В рамках *ER*-модели окружающий мир представляется как набор сущностей (*n*-арных отношений). Полученная семантическая модель предметной области может храниться на компьютере, так же как и сами данные, но, как правило, она содержит общую и конкретную (фактическую) информацию. Семантическая модель вводит термины для описания предметной области и определяет их значения путем задания взаимосвязей и ограничений. Этот уровень представления данных наиболее близок онтологическому представлению.

На основе семантической модели создается логическая схема, описывающая структуры данных в БД, типы данных, взаимосвязи и ограничения. Наиболее популярной и часто используемой для выражения логической схемы является реляционная модель данных. В реляционной модели данные хранятся в таблицах (отношениях), содержащих строки (кортежи), которые, в свою очередь, состоят из ячеек со значениями простых типов данных (целые числа, строки, даты и т.п.). Поэтому для описания логической схемы требуется описать имена таблиц, имена их столбцов (атрибутов) и соответствующие типы данных. Реляционные СУБД требуют, чтобы в каждой таблице был определен набор столбцов (ключ), уникально идентифицирующий строку таблицы. Часто СУБД предлагают средства для поддержания целостности с помощью задания ограничений целостности – утверждений, которые способны отделить корректные состояния базы данных от некорректных.

БД используются для хранения информации о текущем состоянии «окружающего» мира. При этом делается так называемое «предположение о замкнутости». Оно означает, что факт является ложным, пока явно не определен как истинный. Такое предположение работает хорошо, только если БД описывает очень ограниченную область. В частности, в БД не разрешается описывать дизъюнктивную информацию и поддерживается ограниченная форма квантора существования: если нет информации о значении атрибута, то оно считается равным *NULL*.

Декларативный язык *SQL* для описания запросов к реляционным БД и определения структуры таблиц является универсальным и мощным практическим средством. Альтернативу языку *SQL* составляет язык логики предикатов первого порядка, который обладает большими выразительными возможностями и имеет средства для описания знаний, т.е. информации, справедливой для всех или для многих объектов предметной области. Он имеет более «элегантный» вид, если представить, что таблицы являются предикатами.

Например, формула

$$\exists m, d_1, d_2. \text{снабжает} ('intel', r, m, d_1) \wedge \\ \wedge \text{снабжает} ('intel', r, m, d_2) \wedge (d_1 \neq d_2)$$

могла бы определять запрос относительно тех получателей (значения свободной переменной *r*), которые снабжались поставщиком *intel* одним и тем же материалом *m* в различные моменты времени (*d<sub>1</sub>* и *d<sub>2</sub>*).

У пользователя может возникнуть желание получить информацию сразу из нескольких независимых (и скорее всего разнородных) источников. В такой ситуации возникает проблема связывания различных логических схем в одну (предоставляемую пользователю).

Интеграция разнородных источников данных – фундаментальная проблема, возникшая в последние десятилетия перед сообществом разработчиков БД. Цель интеграции данных состоит в том, чтобы предоставить единый интерфейс к различным источникам и позволить пользователям сосредоточиться на определении того, что они хотят узнать. В результате интеграция должна освободить пользователя от поиска релевантных источников дан-

ных, взаимодействия с ними по отдельности, отбора и комбинирования данных из различных источников. Проектирование системы интеграции данных – очень сложная задача.

Рассмотрим «классические» подходы к ее решению. Первый из них состоит в использовании федеративных БД, которые независимо хранят одну и ту же информацию, периодически синхронизируя свои состояния. Для синхронизации  $n$  федеративных БД требуется определить  $O(n^2)$  связей. Другой подход состоит в создании единого централизованного хранилища данных. Данные из разнородных источников периодически копируются в хранилище (требуется  $O(n)$  связей для  $n$  БД). Третий подход (наиболее эффективный, но и трудоемкий) использует технологию создания программных оболочек, или медиаторов (*mediators, wrappers*), обеспечивающих единый интерфейс доступа к различным БД.

Задача «проектирование системы интеграции данных» состоит из нескольких подзадач. Онтологический подход может успешно применяться для решения двух подзадач:

- спецификации содержимого разнородных источников данных в виде онтологии;
- получения ответов на запросы, адресованные интегрирующей системе и основанные на спецификации источников.

## **5.8. Спецификация содержимого разнородных источников данных**

Обычно архитектура системы интеграции данных позволяет явно моделировать данные и информационные потребности (т.е. определять те данные, которые система предоставляет пользователю) на различных уровнях.

Концептуальный уровень содержит концептуальное представление источников и согласованных интегрируемых данных вместе с явным декларативным описанием отношений между их компонентами.

Логический уровень содержит представление источников в терминах логической модели данных.

### **Концептуальный уровень**

Данный уровень содержит формальные описания понятий, отношений между понятиями и дополнительные требования. Эти

описания являются независимыми от системы интеграции и ориентированы на описание семантики приложения. На концептуальном уровне выделяют 3 элемента:

– концептуальная схема уровня предприятия (понимаемого в широком смысле). Здесь представляются наиболее общие понятия, связанные с приложением;

– концептуальная схема информационного источника является концептуальным представлением данных;

– концептуальная схема предметной области используется для описания объединения концептуальной схемы уровня предприятия и различных концептуальных схем информационных источников. Кроме того, сюда же входят дополнительные межсхемные отношения.

Элементарные понятия реляционной модели данных, такие как сущности, отношения и атрибуты, могут быть выражены на языке дескриптивной логики (*DL*) следующим образом. По *ER*-схеме создается база знаний, которая определяется так:

– множество атомарных концептов состоит из множества сущностей и доменных символов;

– множество атомарных отношений получается из множества отношений и атрибутных символов;

– множество аксиом включения состоит из утверждений, которые формализуют иерархические зависимости между сущностями и отношениями, ограничения на сущности, имеющие атрибуты и/или связанные отношениями, и ограничения кардинальности для ролей каждого из отношений.

Важным свойством отображения является его взаимная однозначность, что позволяет строить логические выводы в рамках базы знаний средствами *DL*, а затем переносить результаты на *ER*-модель.

Таким образом, моделирование концептуальных схем БД при помощи онтологий не только возможно, но и дает дополнительные преимущества.

Наиболее интересным элементом на концептуальном уровне является схема предметной области, интегрирующая схему предприятия и схемы информационных источников, а также связывающая схемы информационных источников между собой. При этом используются межсхемные отношения. Они формально представляются как утверждения вида  $L_i \subseteq_{ext} L_j$ ,  $L_i \subseteq_{int} L_j$ , где  $L_i$ ,  $L_j$

являются некоторыми выражениями в различных схемах. Они могут быть либо отношениями с одинаковым числом аргументов, либо концептами. Индекс рядом со знаком включения  $ext$  означает следующее: любой объект, удовлетворяющий выражению  $L_i$ , в  $i$ -м источнике, удовлетворяет также выражению  $L_j$  в  $j$ -м источнике. Например, если известно, что множество сотрудников, описанных в источнике 1, является подмножеством сотрудников, описанных в источнике 2, то это может быть выражено так:

$$\text{Сотрудник}_1 \subseteq_{ext} \text{Сотрудник}_2.$$

Второй тип межсхемных утверждений (соответствует индексу  $int$ ) указывает, что концепт, описанный выражением  $L_i$  в  $i$ -м источнике, является подконцептом того, который описан выражением  $L_j$  в  $j$ -м источнике. Например, если известно, что концепт «сотрудник», описанный в источнике 1, является подконцептом концепта «личность», который описан в источнике 2, то это может быть выражено так:

$$\text{Сотрудник}_1 \subseteq_{int} \text{Личность}_2.$$

### Логический уровень

Данный уровень представляет описание логического содержимого для каждого источника, которое называется схемой источника. Обычно схема источника выражается в терминах набора отношений с использованием логической реляционной модели данных. Связывание логического представления источника и концептуальной схемы предметной области возможно двумя способами:

– подход на основе глобального представления. С каждым концептом концептуальной схемы предметной области ассоциируется запрос над базовыми отношениями источника. Таким образом, каждый концепт можно понимать как представление источника;

– подход на основе локального представления. Каждое отношение в источнике ассоциируется с запросом, описывающим его содержимое в терминах концептуальной схемы предметной области. Другими словами, содержимое каждого базового отношения внутри источника выражается через концепты предметной области.

Для описания содержимого источника используется понятие запрос. Запрос определяется как дизъюнкция конъюнкций над

множеством атомов. Каждый из атомов является концептом, отношением или атрибутом.

### 5.9. Информационный поиск

При упоминании области информационного поиска – *Information Retrieval (IR)* – обычно имеют в виду комплексную деятельность по сбору, организации, поиску, извлечению и распространению информации при помощи компьютерных технологий. Теоретическими и инженерными аспектами реализации этих технологий занимаются соответствующие научные и инженерные дисциплины.

Примерами задач в области информационного поиска [1] являются:

- собственно информационный поиск документов по запросу пользователя;
- автоматическая рубрикация документов по заранее заданному рубрикатору;
- автоматическая кластеризация документов – разбиение на кластеры близких по смыслу документов;
- разработка вопросно-ответных систем – поиск точного фрагмента текста, отвечающего на вопрос пользователя, а не целого документа;
- автоматическое составление аннотации документа и многие другие.

Как любая деятельность, *IR* имеет цель – обеспечить удовлетворение потребности в информации (или информационной потребности) пользователя. Но сама информационная потребность представляет собой некое психологическое состояние человека. Поскольку в настоящее время не существует носителей для такого рода информации, то единственный способ выразить информационную потребность состоит в ее изложении в форме, доступной для обработки машиной – например, в форме запроса, написанного на естественном языке (ЕЯ). Хотя запрос редко в точности соответствует информационной потребности, это единственный способ связи между пользователем и поисковой машиной.

Поисковая машина (поисковик) использует запрос как входные данные для получения того или иного результата – выборки из коллекции документов, соответствующих запросу (поисковая ма-

шина находит документы, релевантные запросу). Здесь возникает вторая проблема: пользователь оценивает результат поиска в соответствии со своей информационной потребностью, а не в соответствии с введенным запросом. В ходе оценки он принимает решение о релевантности (мере соответствия) результата поиска и его (пользователя) информационной потребности. Такую оценку может произвести только сам пользователь. Соответствующее суждение о релевантности и саму релевантность называют истинными. Релевантность, вычисляемая поисковиком на основе его внутренней логики, может не соответствовать истинной релевантности.

С точки зрения пользователя, работа поисковой машины начинается после отправки запроса. Фактически, этому предшествует важный этап индексирования коллекции документов. Он заключается в создании индексных таблиц, значительно ускоряющих обработку запросов. Идея индексирования массивов данных для ускорения доступа применяется повсеместно. Примером индекса может служить алфавитный или предметный указатель в конце книги, оглавление. Особенность индексирования в *IR* состоит в том, что индекс, необходимый для полнотекстового поиска в электронных коллекциях, является наиболее полным. Он должен содержать все термины, которые появляются в документах коллекции. Индекс, который содержит все термины, появляющиеся в документах коллекции, называется обратным (инвертированным) файлом. Часто вместо всевозможных форм каждого слова в инвертированный файл включают только токены – части слов, остающиеся после отсечения окончаний. Например, словам «столы», «столу», «столом» соответствует единственный токен «стол».

Для каждого токена на множество документов вычисляются следующие характеристики:

- число документов, в которых появился данный токен (эта характеристика говорит о распространенности токена в коллекции);
- частота встречаемости токена в коллекции (эта характеристика показывает, насколько данный токен «необычен» по сравнению с другими).

Кроме того, для данной пары «токен – документ (содержащий данный токен)» в инвертированном словаре хранится информация:

- о частоте встречаемости токена в документе;

– о смещении токена от начала документа. Смещение может измеряться в символах или в словах. Эта характеристика часто используется для быстрого извлечения слов контекста и для составления краткого резюме.

Для многих подходов к организации полнотекстового поиска оказывается достаточно инвертированного файла или подобной структуры данных. Но при извлечении релевантных данному запросу документов необходимо определить способ обработки запроса и вычисления значения релевантности для каждой пары «запрос-документ».

### Способы обработки запроса

**Булевский поиск.** Исторически одним из первых способов обработки запросов был так называемый булевский поиск. В этом подходе слова запроса соединяются между собой логическими связками: *and*, *or*, *not*. Допустима группировка при помощи скобок. Таким образом, запрос представляется логической формулой, в которой атомами могут быть термины или какие-либо дополнительные условия. Поисковая машина, основанная на булевом поиске, возвращает документы, для которых формула-запрос принимает истинные значения. Каждому атому формулы сопоставляется множество документов, для которых значение атома истинно. Если атом является термином, то ему сопоставляется множество документов, в которых термин встречается. Затем над множествами выполняются элементарные операции – объединения, пересечения и дополнения, соответствующие логическим связкам между атомами:

$$\begin{aligned} T_1 \vee T_2 &\sim D_{T_1} \cup D_{T_2}, \\ T_1 \& T_2 &\sim D_{T_1} \cap D_{T_2}, \\ \neg T &\sim D_0 / D_T, \end{aligned}$$

где  $T$ ,  $T_1$  и  $T_2$  – атомы;  $D_T$  – множество документов, для которых атом принимает истинное значение;  $D_0$  – множество всех документов коллекции.

Такой подход к обработке запроса имеет ряд недостатков.

На данный запрос поисковая машина может вернуть много документов (или даже все документы коллекции). В этом случае пользователь вынужден последовательно добавлять условия в

запрос, чтобы уменьшить результирующую выборку. Поиск производится методом проб и ошибок.

Как правило, полезную выборку обозримого размера можно получить, задав сложную логическую формулу. При этом от пользователя требуется не только знание правил построения формул, но и достаточно хорошее знакомство с «языком» предметной области.

Вследствие того, что существует только два значения релевантности: «релевантен» (*true*) и «нерелевантен» (*false*), результирующая выборка не может быть упорядочена по релевантности. Все документы одинаково релевантны.

Все атомы формулы имеют одинаковую важность (вес), хотя некоторые из них могут быть «ключевыми», другие – вспомогательными.

Существуют способы улучшения качества булевого поиска. Для автоматического расширения запроса синонимичными терминами можно использовать тезаурус или другой ресурс онтологического характера.

**Ранжированный поиск [64].** Основным способом обработки запросов поисковыми машинами в Интернете является ранжированный поиск. Он основан на вычислении релевантности через распределение частот встречаемости терминов запроса по документам коллекции. На вход может поступать запрос на естественном языке. В процессе предобработки из запроса удаляются стоп-слова (например, «где», «почему» и т.п.) и частицы. Термины сокращаются до токенов. После этого на основе токенов можно было бы автоматически сформировать логическую формулу. Но эксперименты показали, что связывание атомов операцией **AND** дает слишком мало документов в результирующей выборке и многие релевантные документы остаются за ее пределами. Связывание атомов формулы операцией **OR** дает противоположный результат: выборка сильно зашумляется. В данном случае булев подход к обработке естественно-языкового запроса не является адекватным. В этой ситуации дополнительная информация о взаимосвязях терминов (онтология) могла быть использована для формирования более сложной логической формулы.

Вместо того чтобы представлять документы как множества слов, а запрос – как последовательность операций над множествами (как в булевом поиске), предлагается следующее.

Каждый документ коллекции представляется вектором в векторном пространстве, размерность которого равна числу токенов в инвертированном файле. Документ описывается «весами» (координатами) соответствующих токенов. Координатные оси являются попарно ортогональными (токены попарно независимы и образуют базис пространства). Запрос, прошедший предобработку, содержит последовательность токенов и, так же как любой документ, может быть разложен по базису пространства. Далее для вычисления релевантности определяется функция, которая каждой паре векторов ставит в соответствие число из отрезка  $[0,1]$ . Крайние точки соответствуют значениям «нерелевантен» и «полностью релевантен». Промежуточные значения определяют степень релевантности документа запросу (или двух документов коллекции, если требуется найти «похожие» документы). Рассмотрим подробнее, как определяются значения весов документов и функция релевантности.

Пусть  $D = (d_1, \dots, d_N)$  – множество документов коллекции,  $T = (t_1, \dots, t_M)$  – множество токенов. Для каждого фиксированного  $i$  документ  $d_i$  представляется вектором весов

$$W_{ji} = tf_{ji} \cdot idf_{ji}, \quad j=1\dots M,$$

где  $tf_{ji}$  – частота встречаемости токена  $t_j$  в документе  $d_i$  по сравнению с другими токенами документа,  $idf_{ji}$  – величина, обратная частоте встречаемости токена  $t_j$  по всем документам коллекции.

Таким образом, документы коллекции представляются векторами с  $M$  координатами. Запрос тоже может рассматриваться как документ и представляется вектором  $q = (q_1, \dots, q_M)$ . Матрица  $W$ , составленная из векторов документов  $w_{ji}$ , имеет размерность  $M \times N$ . Умножая вектор  $q$  на  $W$  слева, получим вектор  $a = (a_1, \dots, a_N)$ , содержащий значения близости между запросом и всеми документами коллекции. После нормирования  $a_k$ ,  $k=1\dots N$  на модуль вектора  $q$  и вектора  $w_k = (w_{1k}, \dots, w_{Mk})$  вектор  $a$  будет содержать значения релевантности для каждой пары  $(q, d_k)$ . Иными словами, документы коллекции ранжируются по релевант-

ности данному запросу, что полезно при представлении результатов поиска.

**Вероятностная модель [64].** Еще один подход к обработке запроса основан на вероятностной модели. Это попытка описать ранжированный поиск в терминах теории вероятностей. Проблема состоит в том, что частоты, используемые в ранжированном поиске, по своему смыслу не имеют никакого отношения к вероятностям. Число появлений термина в документе не может служить значением случайной величины и использоваться для оценки вероятности появления данного термина в других документах коллекции. Поэтому частоты встречаемости терминов нельзя применять в стандартных формулах теории вероятностей.

В основу модели положен способ вычисления вероятности того, что данный документ релевантен запросу. В случае если вероятность достаточно велика, документ считается релевантным. Основные предположения заключаются в следующем:

- документ  $d$  либо релевантен, либо нерелевантен запросу  $q$  (т.е. для каждого события  $(d, q)$  возможно только 2 элементарных исхода  $(w_0, w_1)$ ;

- определение одного документа как релевантного не дает никакой информации о релевантности других документов.

Таким образом, теория не учитывает ни степень релевантности, ни то, что релевантность одного документа может влиять на релевантность других. Этот способ вычисления релевантности далек и от определения истинной релевантности, и от определения полезной релевантности. Однако сами значения вероятности релевантности могут быть полезны при представлении результатов (для упорядочивания выборки).

Вероятность извлечения из коллекции документа  $D$ , релевантного запросу  $Q$ , может быть выражена так:

$$P(R_Q = X/D), (*)$$

где  $R_Q$  – случайная величина, принимающая значения из множества  $X = \{0, 1\}$ .  $R_Q = 1$  соответствует событию извлечения из коллекции релевантного документа,  $R_Q = 0$  – нерелевантного.

Выражение (\*) для того, чтобы его можно было вычислить, оценивается так:

$$\frac{P(R_Q = 1 | D)}{P(R_Q = 0 | D)} = \frac{P(R_Q = 1)P(D | R_Q = 1)}{P(R_Q = 0)P(D | R_Q = 0)} \approx \frac{P(D | R_Q = 1)}{P(D | R_Q = 0)}.$$

Отношение в левой части равенства («близость» документа и запроса) показывает, насколько вероятность извлечь из коллекции релевантный запросу документ больше вероятности извлечь нерелевантный документ.

Отношение после знака эквивалентности оценивается с использованием распределения терминов запроса по документам коллекции:

$$\frac{P(D | R_Q = 1)}{P(D | R_Q = 0)} \approx \prod_{t \in Q} \frac{P(t | R_Q = 1)}{P(t | R_Q = 0)}.$$

Эксперименты показали, что качество работы поисковых машин на основе вероятностной модели в целом не лучше поисковиков, основанных на ранжированном поиске.

### **Оценка качества информационного поиска**

С ростом числа поисковых машин, различных методик, алгоритмов поиска возникла необходимость сравнивать качество их работы [1, 64]. Для этого были введены две характеристики: точность ( $p$ ) поиска и его полнота ( $r$ ).

Точность (англ. precision, обозн.  $p$ ) – доля релевантных документов выборки по отношению ко всем документам в выборке.

Полнота (англ. recall, обозн.  $r$ ) – доля релевантных документов в выборке по отношению ко всем релевантным документам коллекции.

Эти два критерия обычно конфликтуют. Стопроцентная точность и полнота на практике не достижимы.

Пусть  $N$  – число документов в коллекции,  $n$  – число документов в коллекции, релевантных некоторому запросу,  $m$  – число документов в выборке, полученной системой на данном запросе,  $A$  – число релевантных документов в выборке. Тогда  $p = A/m$ ,  $r = A/n$

Таблица 5.1. Характеристики качества поиска

	<b>Релевантные</b>	<b>Нерелевантные</b>	
Извлечены	<i>A</i>	<i>B</i>	$A+B = m$
Не извлечены	<i>C</i>	<i>D</i>	$C+D = N-m$
	$A+C = n$	$B+D = N-n$	$A+B+C+D=N$

### Попытки улучшить качество информационного поиска на основе онтологических ресурсов

Конкретные примеры использования онтологий для приложений информационного поиска в рамках булевской и векторной моделей реализованы в следующих исследовательских проектах [64]:

- *WordNet* в сочетании с векторной моделью информационного поиска;
- *WordNet* в булевской модели поиска вопросно-ответной системы Южного Методистского университета США;
- Традиционные информационно-поисковые тезаурусы в комбинации с разного рода статистическими моделями;
- Тезаурус для автоматического индексирования в булевских моделях поиска документов, в задаче автоматической рубрикации, автоматического аннотирования;
- Модификация DL – логики для формализации OWL – онтологий, использование признаковых структур для структуризации онтологических описаний [14].

### ***WordNet* в сочетании с векторной моделью информационного поиска**

Лингвистический ресурс *WordNet* [64] разработан в Принстонском университете США. *WordNet* относится к классу лексических онтологий, свободно доступен в Интернете, и на его основе были выполнены тысячи экспериментов в области информационного поиска.

*WordNet* версии 2.1 охватывает приблизительно 155 тысяч различных лексем и словосочетаний, организованных в 117 тысяч понятий, или совокупностей синонимов (*synset*); общее число пар «лексема-значение» насчитывает 200 тысяч.

Основным отношением в *WordNet* является отношение синонимии. Наборы синонимов – синсеты – основные структурные элементы *WordNet*.

Понятие синонимии базируется на критерии, что два выражения являются синонимичными, если замена одного из них на другое в предложении не меняет значения истинности этого высказывания.

В состав словаря входят лексемы, относящиеся к четырем частям речи: прилагательное, существительное, глагол и наречие. Лексемы различных частей речи хранятся отдельно, и описания, соответствующие каждой части речи, имеют различную структуру.

Синсет может рассматриваться как представление лексикализованного понятия (концепта) английского языка.

Считается, что синсет существительных представляет понятия существительных, глаголы выражают глагольные концепты, прилагательные – концепты прилагательных и т.п.

Между существительными в словаре установлены следующие семантические отношения:

- синонимия;
- антонимия;

– гипонимия/гиперонимия – отношение, которое иначе может быть названо ВЫШЕ-НИЖЕ, *is-A* -отношение. Отношение транзитивно и несимметрично. Гипоним наследует все свойства гиперонима. Это отношение является центральным отношением для описания существительных;

– меронимия (отношение ЧАСТЬ-ЦЕЛОЕ). Внутри этого отношения выделяются отношения *быть\_элементом* и *быть\_сделанным из*.

Основным отношением между синсетами существительных является родо-видовое отношение, при этом видовой синсет называется гипонимом, а родовой – гиперонимом. Это транзитивное иерархическое отношение, которое может быть также названо *is-A*-отношением.

Таким образом, отношения между синсетами образуют иерархическую структуру.

При построении иерархических систем на базе родо-видовых отношений обычно предполагается, что свойства вышестоящих понятий наследуются нижестоящими – так называемое свойство наследования.

В *WordNet* выделяются три подвида отношения ЧАСТЬ-ЦЕЛОЕ: собственно часть, *быть\_элементом* и *быть\_сделанным\_из*.

Таким образом, ворднеты – это сеть языково-специфичных лексикализованных единиц (в отличие от формальных онтологий, которые представляют собой структуру данных с формально определенными понятиями).

Основные предполагаемые применения ворднетов – это предсказание той или иной возможной замены лексических единиц в тексте для целей информационного поиска, генерации текстов, машинного перевода, разрешения лексической многозначности.

Для того чтобы установить связи между различными языками, в проекте *EuroWordNet* синсеты каждого ворднета имеют ссылку на так называемый межязыковой индекс (*interlingual index*), в качестве которого выбираются синсеты Принстонского *WordNet*.

### **WordNet: применение в информационном поиске**

Для того чтобы попытаться реализовать схему автоматического концептуального индексирования [64] и концептуального поиска, необходимо иметь лингвистический ресурс, организованный на основе понятий или значений слов. Поэтому такие ресурсы, как *WordNet*, могут использоваться как база для организации приложений концептуального индексирования и поиска.

#### **Векторная модель информационного поиска с вектором по синсетам WordNet**

Целью экспериментов была попытка выполнить поиск документов на основе не отдельных слов, а значений *WordNet*. Для каждого документа сначала выполняется процедура разрешения многозначности существительных, которая выбирает единственное значение и в результате которой каждому тексту ставится в соответствие вектор синсетов *WordNet*. После того как вектор создан, с ним могут выполняться такие же операции, как и с пословными векторами.

Эффективность использования векторов синсетов сравнивалась с эффективностью информационного поиска на основе стандартной модели, использующей вектора слов.

Оценки эффективности информационного поиска на основе показателя средней точности показали серьезное ухудшение эффективности для векторов, включающих синсеты (от 6,2 до 42,3 %).

### **Эксперименты по расширению запросов на основе отношений *WordNet***

Другая группа экспериментов по использованию *WordNet* в информационном поиске исследовала возможность расширения запросов синонимами или другими словами, связанными со словами запроса отношениями, которые описаны в *WordNet*. Для экспериментов были использованы следующие соображения.

Во-первых, расширяться должны только важные для запроса понятия. Важность аппроксимируется количеством документов, в которых встречается конкретное слово запроса – слова, частотность которых в документах коллекции больше некоторого числа  $N$ , не участвуют в расширении запроса.

Исследовались различные величины  $N$  – 10 % коллекции и 5 % коллекции.

Для расширения запроса использовались синсеты, находящиеся на расстоянии одного или двух отношений от исходных синсетов – все виды связей трактовались одинаково.

Добавленные слова могли учитываться с разными величинами весов:  $w = 0,3, 0,5, 0,8$ .

Максимальное улучшение, которое удалось получить, – 0.7 % средней точности, что не является статистически значимой величиной ( $N = 5\%$ , расстояние – 2,  $w = 0.3$  ).

Авторы подчеркивают, что идея аппроксимации разрешения многозначности путем поиска повторов в списках расширения оказалась неэффективной ввиду того, что чаще всего этот метод приводил к добавлению в запрос очень общих слов, таких как «система».

Выводы авторов эксперимента заключаются в том, что для успешного применения *WordNet* в информационном поиске необходимо значительно улучшить эффективность автоматического разрешения лексической многозначности, между тем парадигматических отношений в *WordNet* недостаточно для решения этой задачи.

## Проект *Meaning*

В проекте планируется выполнить три последовательных цикла масштабного разрешения лексической многозначности и извлечения знаний для пяти европейских языков, включая баскский, испанский, итальянский, голландский и английский языки. Накопленные знания должны храниться в Многоязычном Центральном Репозитории.

### Эксперименты по семантическому индексированию в рамках проекта *Meaning*

В рамках европейского проекта *Meaning* голландская компания *Irion Technologies* разработала технологию концептуального индексирования TwentyOne, комбинирующую лингвистический и статистический подходы. Основой технологии является статистическая машина поиска, базирующаяся на стандартной векторной модели и обеспечивающая быстрый поиск документов.

Лингвистические технологии используются для улучшения результатов, выданных статистической машиной, в двух направлениях:

- максимизация полноты результатов за счет использования синонимии ворднетов;
- максимизация точности результатов за счет сравнения запросов с конкретными фразами документов, а не с целыми документами.

Суть технологии в том, что сначала выдаются документы, которые имеют наибольшее совпадение по концептам фраз с запросом. Среди документов, имеющих одинаковое количество сопоставленных понятий между собственными фразами и запросом, первыми выдаются наиболее схожие по конкретному набору слов.

В проводимых экспериментах для сравнения были построены четыре индекса:

- *HTM* – традиционный пословный индекс;
- *NP* – индексы именных групп из запроса, с применением пословных методов, без использования ворднетов;
- *FULL* – полные индексы с использованием ворднетов, но без процедуры разрешения многозначности, что приводит к полному расширению по синонимам и переводам для всех возможных значений слов запроса;

– *WSD* – индексы, использующие ворднеты вместе с процедурой снижения многозначности на основе предметных областей ворднет.

В эксперименте индексы тестились в системе автоматической рубрикации текстов на коллекции *Reuter*. Описано, что максимальных значений *F*-меры система автоматической рубрикации достигает для индекса *WSD*: полнота – 80,7, точность – 72,2. Минимум системы имеет на базе индекса *HTM*: полнота – 67,8, точность – 70,4.

### **WordNet: применение в вопросно-ответных системах**

Вопросно-ответная система представляет собой вид информационно-поисковых систем. Она должна предоставить не набор документов, которые наиболее релевантны поставленному вопросу, а выдать точный ответ на данный вопрос.

С 1999 года проводится соревнование по вопросно-ответным системам в рамках конференции *TREC*, с 2003 года соревнования вопросно-ответных систем в многоязычном контексте начаты на конференции *CLEF*.

Одной из самых эффективных систем в вопросно-ответной секции конференции *TREC* стала вопросно-ответная система Южного Методистского университета США, которая на нескольких этапах обработки вопроса и поиска ответа обращается к информации, хранимой в тезаурусе *WordNet*. В разработанной системе *WordNet* используется для:

- распознавания типа вопроса;
- классификации типов ответов;
- реализации лексических и семантических замен.

Лексические и семантические замены осуществляются в момент сопоставления формальной структуры вопроса и ответа. Поиск в системе организован на основе обработки булевых запросов.

### **Представление толкований *WordNet* в виде логических выражений: проект *eXtended WordNet***

Многие исследователи отмечают нехватку информации, описанной в *WordNet*, для различения значений, в нем перечисленных.

В рамках проекта *eXtended WordNet* разработчики предполагают, что важным источником дополнительной информации могут стать толкования, приписанные к синсетам *WordNet*. Для того

чтобы эти толкования можно было использовать в автоматических режимах компьютерных приложений, необходимо каждому знаменательному слову толкований сопоставить его значение-синсет и представить это толкование в виде формализованного выражения.

### **Информационно-поисковые тезаурусы**

Информационно-поисковый тезаурус (ИПТ) [64] – это контролируемый словарь терминов на естественном языке, явно указывающий отношения между терминами и предназначенный для информационного поиска.

Основными целями разработки традиционных ИПТ являются следующие:

- обеспечение перевода естественного языка документов и пользователей на контролируемый словарь, применяемый для индексирования и поиска;
- обеспечение последовательного использования единиц индексирования;
- описание отношений между терминами;
- использование как поискового средства при поиске документов.

Разработчики тезаурусов предполагают, что понятие предметной области обычно имеет несколько возможных вариантов лексического представления в тексте, которые рассматриваются как синонимы. Среди таких синонимов выбирается *дескриптор* – термин, который рассматривается как основной способ ссылки на понятие в рамках тезауруса. Другие термины из синонимического ряда, включенные в тезаурус, называются *аскрипторами* или недескрипторами. Они используются как вспомогательные элементы, текстовые входы, помогающие найти подходящие дескрипторы.

Дескрипторы тезауруса должны соответствовать выбранной предметной области тезауруса. Каждый дескриптор, внесенный в тезаурус, должен представлять отдельное понятие данной области. Дескриптор может быть однословным или многословным. Поскольку часто бывает достаточно трудно понять, представляет ли отдельное понятие многословное словосочетание, многие тезаурусы и руководства уделяют особое внимание основным

принципам включения в тезаурус в качестве дескрипторов многословных терминов.

Основными типами отношений, обычно отражаемых в информационно-поисковых тезаурусах, являются следующие:

- род – вид;
- часть – целое;
- причина – следствие;
- сырье – продукт;
- административная иерархия;
- процесс – объект;
- функциональное сходство;
- процесс – субъект;
- свойство – носитель свойства;
- антонимия.

Такие содержательные типы связей между дескрипторами чаще всего не отражаются в подробном перечне отношений тезауруса, а записываются с помощью небольшого набора отношений, которые обычно разделяются на два типа: иерархические и ассоциативные.

Иерархические отношения обладают свойствами транзитивности и антисимметричности, которые могут быть использованы при избыточном индексировании в интересах повышения эффективности информационного поиска. Предпочтительно указывать связи между дескрипторами как отношения иерархического вида, если они обладают этими свойствами. Применяемые в ИПТ иерархические отношения могут дифференцироваться на отдельные виды.

Основным иерархическим отношением, используемым в ИПТ, является родовидовое отношение (оно же – отношение НИЖЕ-ВЫШЕ). Родовидовая связь устанавливается между двумя дескрипторами, если объем понятия нижестоящего дескриптора входит в объем понятия вышестоящего дескриптора.

Также в качестве иерархического отношения в ИПТ может устанавливаться отношение ЧАСТЬ-ЦЕЛОЕ.

Отношение ассоциации является неиерархическим и ассоциативным. Ассоциативное отношение наиболее трудно определить. Российский стандарт на создание ИПТ указывает, что «ассоциативное отношение является объединением отношений, не входя-

щих в иерархические отношения или в отношения синонимии. Допускается включать в ассоциативное отношение все виды отношений, кроме синонимии и отношения РОД-ВИД».

Одним из подходов для автоматизации индексирования по традиционным ИПТ является подход, основанный на правилах. Такой подход к автоматическому индексированию был реализован по тезаурусу EUROVOC.

Правила могут быть простыми и сложными. Простые правила не содержат условий. Сложные правила содержат такие условия, как Близость (на расстоянии трех слов по тексту, в одном предложении, в том же самом поле, например, в поле реферата), Местонахождение (в заголовке, в тексте реферата или документа, в начале предложения, в конце предложения), Формат (с большой буквы, все большими буквами). Всего было создано около 40 тысяч правил.

В качестве других подходов автоматизации индексирования используются статистические методы.

При использовании вышеуказанных методов большинство автоматически присвоенных дескрипторов выглядят весьма релевантными тексту документа.

## 5.10. Семантический поиск

Одна из наиболее важных задач, которую можно решить, используя онтологии – это семантический поиск [1, 12, 13]. В настоящее время проблема поиска информации в больших массивах усугубляется еще и тем, что существующие поисковые механизмы осуществляют поиск информации без учета семантики слов, входящих в запрос, а также контекста, в котором они используются.

Благодаря онтологиям, появилась возможность создания семантических сетей.

В основе семантической сети лежат три принципа: агрегация, безопасность и логика. Агрегация означает совместное использование данных. Для этих данных создается соответствующая семантическая информация (онтологии), позволяющая использовать их надлежащим образом. Логика – это набор правил описания информационной структуры данных, протоколы и язык описания страниц. Именно логика дает семантической сети правила вывода

для проведения рассуждений и методики выбора тактик выполнения операций с данными, чтобы получить ответы на вопросы.

В основу безопасности, обеспечивающей доверие к семантической сети, положены цифровые подписи, которые могут использоваться агентами и компьютерами для проверки того, что информация получена из достоверного источника, например, от какого-то публичного сервиса или персонального агента другого доверенного пользователя.

Важнейшую роль в семантической сети (рис. 5.5) должны играть специальные программы – интеллектуальные агенты, в задачу которых входит работа с информацией, представленной в семантической сети. Агенты по заданиям пользователей будут находить источники информации, запрашивать данные, сопоставлять и проверять их на соответствие критериям поиска, а затем выдавать ответ в удобной для пользователей форме.

Агенты будут способны обмениваться между собой не только информацией и правилами логических выводов, используемых в онтологиях, но и цепочками построенных ими рассуждений, чтобы пользователь мог при необходимости проверить результат, либо на основании уже собранной информации другой агент попробовал найти более оптимальное решение или уточнить какие-то условия первоначального запроса. В определенных ситуациях для решения поставленной задачи может потребоваться и передача одним персональным агентом другому агенту некоторой личной информации пользователя (рис. 5.6).



**Рис. 5.5.** Поиск нужной информации через поисковый сервис, работающий с семантической сетью

Семантическая сеть должна стать надстройкой над *WWW*. С ее помощью станет возможным создание новых сервисов, которые технически невозможны при нынешних принципах организации и работы Сети. Цепочки логических рассуждений, осуществляемые

агентами, позволяют получать информацию, представленную в Сети, в разрозненном виде, то есть из разных информационных источников. В дополнение к поисковым движкам (*search engine*) поисковые системы обзаведутся специальными логическими движками (*logical engine*), предназначенными для поиска и обработки информации в семантической сети. Комбинация этих двух поисковых механизмов позволит поднять качество и точность поиска в Сети на совершенно иной уровень (рис. 5.7).

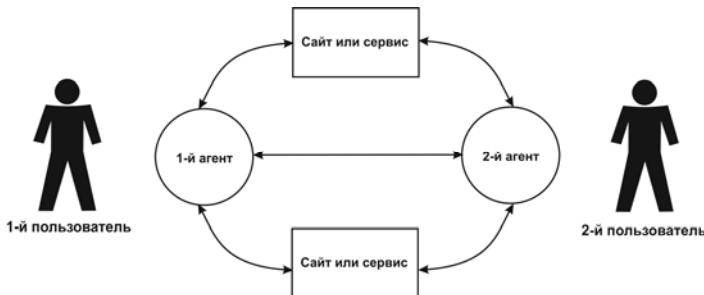


Рис. 5.6. Взаимодействие агентов

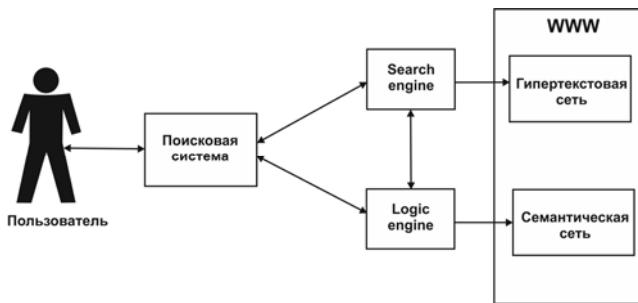


Рис. 5.7. Принцип работы поисковой системы, использующей поисковый и логический модули

По всем расчетам, переход к семантической сети не будет ни простым, ни быстрым. Работы начались еще в 1998 году, а официальная рекомендация *OWL* консорциумом *W3C* в качестве основного языка для описания онтологий была представлена только в феврале 2004 года. Немало специалистов считают, что семантический *Web* – это утопия, и с ними можно частично согласиться, поскольку его внедрение требует немалых усилий от различных категорий разработчиков и пользователей, включая разработчиков

крупных информационных порталов, поисковых систем, каталогов и небольших сайтов, а также от разработчиков интеллектуальных программ-агентов.

Чтобы разработчики сайтов начали повсеместно внедрять поддержку онтологий в свои ресурсы, у них должен быть стимул, то есть использование онтологий должно давать их сайту некоторые преимущества, например, увеличение числа переходов с поисковиков или увеличение числа пользователей за счет предоставления дополнительных сервисов и т.п. Необходимы мощные и гибкие программы-агенты, которые смогут полноценно использовать возможности семантического *Web'a*. Но их активная разработка начнется только тогда, когда у пользователей появится реальная необходимость в них.

### 5.11. Онтологии в электронной коммерции

Приобретения товара через Интернет-магазины дает многие преимущества, такие как: облегчает выбор товара, предоставляет более конкретную информацию о продаваемых товарах, упрощает покупку товара.

Онтология – это способ формализации области знаний о продаваемом товаре. Она описывает иерархию товаров, их свойства, их связь с характеристиками.

Создается web-сервис выбора товара [52, 66, 68], реализованный с помощью применения технологий *Semantic Web*. Функционирование web-сервиса основано на использовании онтологии, содержащей иерархию и характеристики товара, а также их свойства и типы. Технология «связывания» онтологии и web-сервиса реализуется с помощью *SPARQL*-запросов. *SPARQL*-запросы [46, 48] – это запросы к данным, представленным по модели *RDF*, а также протокол для передачи этих запросов и ответов на них. Под *RDF* мы понимаем формат данных (в виде ориентированного маркированного графа) для представления информации во всемирной паутине.

Онтология сегодня играет ключевую роль в системах электронной коммерции категории *business-to-business*. Специалисты-онтологи гарантируют, что деловые партнеры будут использовать один и тот же язык для общения, формирования заказов и поставок товаров и услуг через *Internet*. Необходимо структурировать

большие объемы информации, тем самым помогая клиентам находить нужные продукты и принимать решение о покупке, а для этого требуется целостное описание информации, хранящейся в электронных каталогах.

Онтология становится ключевой функцией многих проектов в области электронного бизнеса: компания *Yahoo!*, к примеру, держит целый штат онтологов, называемых здесь «серферами» (*surfer*) – специалистами по организации поиска и обмена информацией. Эти люди принимают участие в разработке программного обеспечения, которое играет существенную роль в организации взаимодействия с пользователями и может применяться при создании текстовых процессоров и автономных агентов.

### **5.12. Примеры существующих проектов и систем**

Можно привести некоторые примеры существующих систем, содержащих онтологические приложения.

В сфере *информационного поиска* заслуживает упоминания европейский исследовательский проект под названием *CROSS-MARC*. Участники этого проекта делают упор на необходимости широкого использования онтологии для разделения отраслевых и общепонятийных знаний, считая, что это облегчит извлечение информации из различных источников, сузит поисковые запросы и улучшит качество выдаваемых результатов. Эта задача оказывается смежной с задачей автоматической рубрикации текстов, в ходе которой производится распределение текстов по рубрикам на основе автоматических методов и использования онтологии.

Онтологии могут также лежать в основе различных *вопросно-ответных систем* и способствовать улучшению анализа запросов и точности ответов. Можно привести пример демонстрационной вопросно-ответной системы *YAWA*. *YAWA* по запросу выдает информацию о главах государств и правительствах стран мира. Она обладает сведениями о том, кто занимает указанную должность в данной стране. Кроме этих связей, в нее заложены знания о соотношении названия и основной функции (глава государства и/или правительства) высших государственных должностей в отдельно взятой стране в зависимости от типа системы государственного управления. Таким образом, при выдаче ответов по запросу система учитывает заложенные в нее сведения об окружающей

действительности: набор понятий, отношений между ними, ограничений на отношения и список конкретных экземпляров.

Существует большое количество *проектов в области медицины*, использующих онтологии в своих приложениях. Так, можно привести пример проекта *MuchMore*, являющегося частью *InfoMap* и занимающегося разработкой методов организации информации на различных языках и в частности медицинской области знания. Их исследование основывается на использовании иерархии понятий для предметных областей, и, следовательно, технологиях извлечения многоязычных терминов и отношений. Их продукт помогает осуществлять поиск документов на различных языках по медицинской области знания. Медицинская область знания очень перспективна в этой сфере, так как для нее уже создано большое количество онтологий и структурированных источников знания, а также присутствует множество текстов в данной области, требующих обработки. Это тексты, описывающие карты больных, случаи заболеваний, общие описания разных болезней и многие другие. Проект *MuchMore* помогает выстроить взаимосвязи между всеми типами текстов в данной области.

В качестве примера практического использования *онтологических моделей технологий* можно привести систему *ONTOLOGIC*, предназначенную для создания и поддержки распределенных систем нормативно-справочной информации (НСИ), ведения словарей, справочников и классификаторов и поддержки системы кодирования объектов учета (рис. 5.8). Онтология обеспечивает непротиворечивое накопление любого количества информации в стандартной структуре классификации. Такой подход гарантирует однозначную идентификацию ресурсов независимо от различных трактовок их наименований разными производителями. При использовании данной системы осуществляется эффективный контроль и верификация данных, проверки корректности, полноты и непротиворечивости данных как на этапе анализа и нормализации существующих данных, так и при занесении новых элементов данных.

Основу системы составляет технологическая среда для постоянного, в режиме реального времени, взаимодействия пользователей: потребителей информации (сотрудников служб и функциональных подразделений) и экспертов, отвечающих за ведение нормативно-справочной информации.

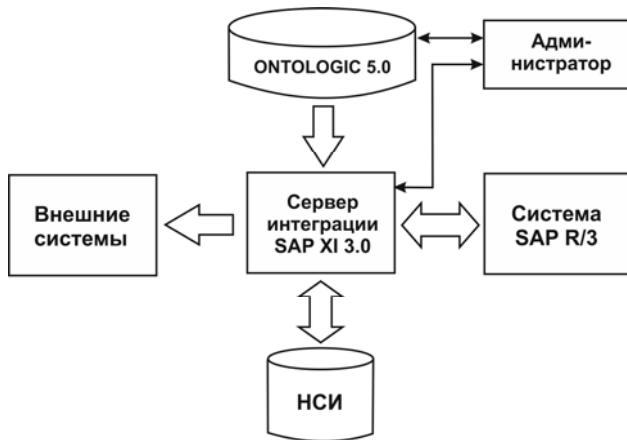


Рис. 5.8. Интеграция приложений в *ONTOLOGIC*

Для обеспечения однозначной идентификации и классификации объектов в системах НСИ разработана методика, использующая онтологическую модель формального описания классифицируемых данных, обеспечивающая выделение ключевых свойств объектов классификации и построение на их основе классификационного кода. Выделяются классы (группы однородной продукции) по принципу однородности набора технических и потребительских характеристик, и для каждого материала формируется классификационный код, включающий код класса и коды всех свойств и их значений для данного материала.

Онтология обеспечивает непротиворечивое накопление любого количества информации в стандартной структуре классификации. Такой подход гарантирует однозначную идентификацию ресурсов независимо от различных трактовок их наименований разными производителями.

Данная технология предусматривает создание типового решения для управления основными данными и НСИ для промышленных предприятий, холдингов и государственных структур. В качестве технологической платформы используется *SAP MDM* (*Master Data Management*), предназначенный для интеграции различных (в том числе разноплатформенных) приложений в масштабе компании, холдинга, отрасли, госструктуры и т.п., а также для организации и управления отраслевой или корпоративной нормативно-справочной информацией (мастер-данными).

Следующий пример – это проект ***TOVE*** (*Toronto Virtual Enterprise*). Цель проекта – создание модели данных, которая должна:

- обеспечить общую терминологию для предметной области, приложения которой могут совместно использоваться и пониматься каждым участником общения;
- дать точное и по возможности непротиворечивое определение значения каждого термина на основе логики первого порядка;
- обеспечить задание семантики с помощью множества аксиом, которые автоматически позволяют получать ответ на множество вопросов о предметной области.

***TOVE*** должен обеспечить построение интегрированной модели некоторой предметной области, состоящей из следующих онтологий: операций, состояний и времени, организации, ресурсов, продуктов, сервиса, производства, цены, количества.

***Ontolingua*** – система, разработанная в Стэнфордском университете, которая обеспечивает распределенную совместную среду для просмотра, создания, редактирования, модификации и использования онтологии. Сервер системы поддерживает до 150 активных пользователей, некоторые из которых дополняют систему описанием своих проектов.

Среди множества других проектов ***Ontolingua*** использует проект *Enterprise project*.

***Enterprise Project***. Целью проекта является улучшение (где необходимо, замена) существующих методов моделирования при помощи набора средств, позволяющих интегрировать различные методы и средства моделирования предприятия. Предполагается создание таких инструментальных средств, которые обеспечивают: фиксирование и описание конкретной предметной области; определение задач и требований (согласующихся с онтологией); определение и оценку вариантов решений и альтернативных проектов, реализацию выбранной стратегии.

При независимой разработке инструментальных средств возможно использование разной терминологии, что может привести к конфликтам и неоднозначности при их интегрировании. Для решения этой проблемы была построена онтология, в которой задан набор часто используемых и общепринятых терминов, таких как деятельность, процесс, организация, стратегия, маркетинг.

**KACTUS.** Цель проекта – построение методологии *многократного применения знаний о технических системах во время их жизненного цикла*. Это необходимо, чтобы использовать одни и те же базы знаний для проектирования, оценки, функционирования, сопровождения, перепроектирования и обучения.

*KACTUS* поддерживает интегрированный подход, включающий производственные и инженерные методы и методы инженерии знаний, на основе создания онтологической и вычислительной основы для многократного использования полученных знаний параллельно с различными приложениями технической области. Это достигается при помощи построения онтологии предметной области и ее многократного использования в различных прикладных областях. Кроме того, делается попытка объединить эти онтологии с существующими стандартами (например, *STEP*), применяя онтологии там, где возможно фиксирование данных о конкретной области.

Основным формализмом в *KACTUS* является *CML (Conceptual Modeling Language)*.

Инструментарий *KACTUS* представляет собой интерактивную среду, в которой можно экспериментировать с теоретическими результатами (организовывать библиотеки онтологии, преобразовывать данные между онтологиями, делать преобразования для различных формализмов), а также осуществлять практические действия (просмотр, редактирование и уточнение онтологии в разных формализмах).

**OntoSeek** – информационно-поисковая система, которая разработана для семантически ориентированного поиска информации, комбинируя управляемый онтологией механизм установления соответствия смысла и мощные системы моделирования.

**SHOE** (*Simple HTML Ontology Extensions*) позволяет авторам аннотировать свои Web-страницы, внося в них семантическое содержание. Основным компонентом *SHOE* является онтология, которая содержит информацию о некоторой области. Используя эту информацию, средства поиска и построения запросов обеспечивают более релевантный ответ на запрос по сравнению с существующими поисковыми машинами, так как предоставляется возможность включать в Web-страницы знания, которые интеллектуальные агенты могут действительно прочитать. Для этого *SHOE*

дополняет *HTML* набором специальных тэгов для представления знаний. *SHOE* позволяет находить знания с помощью таксономии и правил вывода, существующих в онтологии.

*Plinus*. Целью проекта является  *полуавтоматическое извлечение знаний из текстов на естественном языке*, в частности, литературы о механических свойствах керамических материалов. Так как тексты охватывают широкий диапазон понятий, требуется множество интегрированных онтологий для охвата таких понятий, как керамические материалы и их свойства, способы их обработки, различные дефекты материалов, например, такие как трещины и поры.

В работе [5] для решения задачи повышения эффективности поиска в сети Интернет предлагается строить  *порталы знаний*, каждый из которых предоставляет доступ к ресурсам сети Интернет определенной тематики. Основу таких порталов знаний составляют онтологии, содержащие описание структуры и типологии соответствующих сетевых ресурсов.

Интересное применение онтологий реализовано в ДО РАН [70]. Специалистами была построена «медицинская» онтология, позволяющая делать выводы. Задав симптомы, с помощью онтологий можно вывести диагноз. Еще одно применение описано в статье [6], «использование онтологии для построения инновационных цепочек в системе поддержки инновационной деятельности в регионе». Система реализуется в виде Интернет-портала и включает в себя, с одной стороны, информационную систему со средствами создания и интеграции связанных с инновациями разнородных информационных ресурсов, а с другой, – развитые средства персонального участия в инновационной деятельности специалистов различного профиля. Важным компонентом, обеспечивающим интеллектуализацию таких рабочих мест, является механизм, поддерживающий интерактивное построение инновационных цепочек. Создание цепочек выполняется по автоматически генерируемому сценарию, структура которого определяется структурой инновационной цепочки, заданной в онтологии инновационной деятельности и видом инновационного запроса.

В [13] рассматривается подход к интеллектуализации *систем документооборота*, основанный на использовании знаний о предметной области, лингвистическом анализе текста документов и его содержательном индексировании.

Интересный подход описан в [54], где строится общая онтология тезаурусов (*Generic Thesaural Ontology*) *GTO*. В статье представляется подход к описанию различных тезаурусов на одном «общем» языке. Описан способ унификации шести основных тезаурусов их моделей, классов и отношений в абстракции более высокого уровня. Эти тезаурусы были выбраны, так как они хорошо известны, используются различными сообществами в различных областях, представленные как функциональной, так и субъектной схемой и основываются на стандарте *ISO 2788*.

Вот эти тезаурусы:

- *Australian Government Thesaurus [Keyword AAA]*
- *Getty Art and Architecture Thesaurus [AAT]*
- *Getty Thesaurus of Geographic Names [TGN]*
- *Library of Congress Subject Headings [LCSH]*
- *OCLC Dewey Decimal Classification [OCLC]*
- *Medical Subject Headings [MeSH]*

### **Общепромышленный онтологический классификатор [58].**

В настоящее время отсутствие кодификатора является одной из главных сдерживающих причин при внедрении федеральной и региональных систем электронных госзакупок. А поскольку создание таких систем предусмотрено ФЦП «Электронная Россия», проблема кодификации является делом государственным.

Одним из вариантов решения проблемы является использование общепромышленного онтологического классификатора, созданного в НЦИТ «Интертех».

По мнению вице-президента «Интертех» Андрея Гребнева, эта разработка лишена большинства недостатков существующих классификаторов благодаря соединению онтологического способа описания продукции и современной программной среды. С 2002 года этот классификатор используется на электронной торговой площадке «Инмарсис», в базе данных которой содержится более 1,5 млн наименований продукции, с 2003 года входит в состав электронной системы снабжения и используется как основа корпоративных классификаторов в ряде компаний.

В основе собственно кодификатора лежит онтологическая модель, в которой иерархическая структура сочетается с выделением базовых свойств и характеристик объектов. При реализации этой модели в классификаторе формируется кодификатор, в котором

происходит объединение продукции в иерархические группы (классы) и присвоение каждой позиции уникального кода, учитываяющего местоположение класса в иерархии, свойство и значение свойства. Для удобства поиска продукции классификатор содержит навигаторы (иерархические структуры, соответствующие функциональным задачам различных групп пользователей), количество которых (как и число пользователей онтологического классификатора) неграничено.

Онтологический классификатор реализуется в *Web*-среде, прост в использовании и обладает интуитивно понятным интерфейсом.

***Dbpedia***. *Dbpedia* – проект, направленный на извлечение структурированной информации из данных, созданных в рамках проекта Википедия. *Dbpedia* позволяет пользователям запрашивать информацию, основанную на отношениях и свойствах ресурсов Википедии, в том числе ссылки на соответствующие базы данных. *Dbpedia* был назван Тимом Бернерсом-Ли одним из наиболее известных примеров использования связанных данных [3].

Проект был начат группой добровольцев из Свободного университета Берлина и Лейпцигского университета, в сотрудничестве с *OpenLink Software*[3], и впервые был опубликован в 2007 году.

Статьи из Википедии в основном состоят из свободного текста, но также включают такие виды структурированной информации, как шаблоны, категории, изображения, географические координаты и ссылки на внешние веб-страницы. Эта структурированная информация извлекается и формируется в виде единой базы данных, доступной по запросу. Данные *Dbpedia* доступны на условиях свободной лицензии.

По состоянию на сентябрь 2011 года, базы данных *Dbpedia* описывают более 3,64 млн понятий, из которых 1,83 млн классифицированы в соответствии с онтологией, в том числе 416 000 персоналий, 526 000 мест, 106 000 музыкальных альбомов, 60 000 фильмов, 17 500 видеоигр, 169 000 организаций, 183 000 биологических видов и 5 400 заболеваний. Метки и тезисы содержатся для 3,64 млн понятий на 97 языках; 2 724 000 ссылок на изображения и 6 300 000 ссылок на внешние веб-страницы; 6 200 000 внешних ссылок на другие базы данных *RDF*-формата, 740 000 категорий Википедии [11].

Проект *Dbpedia* использует *Resource Description Framework (RDF)* для представления извлеченной информации. По состоянию

на апрель 2010 года, базы данных *DBpedia* состоят из более чем 1 млрд единиц информации, из которых 257 млн были взяты из английской версии Википедии и 766 млн извлечены из версий на других языках.

Одна из проблем при извлечении информации из Википедии состоит в том, что одни и те же понятия могут быть выражены в шаблонах разными способами, например, понятие «место рождения» может быть сформулировано в английском языке как «*birthplace*» и как «*placeofbirth*». Из-за этой неоднозначности запрос проходит по обоим вариантам для получения более достоверного результата. Для облегчения поиска при сокращении количества синонимов был разработан специальный язык – *DBpedia Mapping Language*, а у пользователей *Dbpedia* появилась возможность повысить качество извлечения данных с помощью сервиса *Mapping*. В связи с большим разнообразием шаблонов и свойств, используемых в Википедии, дальнейшая разработка этого языка открыта для сотрудничества.

*DBpedia* извлекает фактическую информацию со страниц *Wiki-pedia*, позволяя пользователям найти ответы на вопросы в ситуациях, когда требуемая информация находится в нескольких различных статьях Википедии. Запрос осуществляется с помощью *SPARQL* (*SQL*-подобный язык запросов для *RDF*). Например, вы интересуетесь японской Сёдзё манга *Tokyo Mew Mew*, и хотите найти другие работы иллюстратора этой сёдзё. *DBpedia* объединяет информацию из записей в Википедии в статьях *Tokyo Mew Mew*, *Mia Ikumi*, а также *Super Doll Licca-chan* и *Koi Cupid*. *DBpedia* сводит полученную информацию в единую базу данных, и при следующем запросе нет необходимости уточнять, где именно расположен фрагмент требуемой информации:

```
PREFIX dbprop: <http://dbpedia.org/property/>
PREFIX db: <http://dbpedia.org/resource/>
SELECT ?who ?WORK ?genre WHERE {
    db:Tokyo_Mew_Mew dbprop:illustrator ?who.
    ?WORK dbprop:author ?who.
    OPTIONAL { ?WORK dbprop:genre ?genre } .
}
```

## Краткая информация о проекте *DBpedia*

Тип продукта  
Разработчик

Семантическая паутина, связанные данные  
Лейпцигский университет,

Языки интерфейса	Свободный университет Берлина, <i>OpenLink Software</i>
Первый выпуск	<i>Scala, Java, Virtuoso Server Pages</i> 23 января 2007
Аппаратная платформа	<i>Virtuoso Universal Server</i>
Последняя версия	<i>DBpedia 3.5</i> (12 апреля 2010)
Лицензия	<i>GNU General Public License</i>
Сайт	<i>dbpedia.org</i>

### **Контрольные вопросы**

1. Назовите основные области применения онтологий.
2. Как онтологии используются в системах искусственного интеллекта?
3. Каковы основные идеи Semantic Web?
4. Какие средства описания семантики предметной области получили развитие в последние годы?
5. Как онтологии используются в концептуальном моделировании?
6. Какова роль онтологий в интеграции разнородных источников данных?
7. Как онтологии используются при информационном поиске?
8. Что Вы знаете об онтологическом ресурсе WordNet?
9. Какова роль онтологии при взаимодействии агентов?
10. Как онтологии используются в электронной коммерции?

### **Контрольные задания**

Подготовить презентацию и реферат по согласованной с преподавателем теме. Тему выбрать из списка, приведенного в разделе «Основные области применения онтологий».

## **Заключение**

Подготавливая это пособие, автор ставил перед собой цель собрать воедино имеющуюся в настоящий момент информацию об онтологиях предметных областей и способах их разработки. Пособие содержит как практический материал по созданию онтологии в редакторе Protege 4, так и теоретический – о формализмах, лежащих в основе онтологий, языках их описания и о способах вывода новых знаний.

Что же такое онтология и зачем она нужна потребителю информационных услуг?

Человек отображает предметную область в своем сознании в виде слов, понятий, концептов и категорий. Онтология – это выражение человеческого познания предметной области, следовательно, она оперирует словами, понятиями, концептами и категориями, которые являются наименованиями сущностей реального мира. Построить онтологию значит построить иерархию понятий, концептов, категорий. Онтология – это терминология для обмена знаниями между исследователями в определенной предметной области.

Для потребителя информационных услуг онтология – это классификатор: разбивает объекты (индивидуальности) по группам в соответствии с определенными признаками, что позволяет одну и ту же информацию организовывать различными способами в соответствии с потребностями пользователя. Эта возможность полезна при разработке сайтов и интерфейсов приложений, работающих с базами данных.

В то же время онтология это средство обобщения, так как класс обобщает все подчиненные ему концепты, что может быть использовано при информационном поиске.

Онтология дает возможность получать путем вывода новые знания. В сокращенном виде передает смысл (логическую канву) информации. Может из разных представлений выделять общее необходимое потребителю знание, отвечающее его потребностям.

Изнутри онтология – это база знаний, включающая в себя утверждения общего характера ТВох о предметной области и утверждения частного характера АВох об индивидуальностях. Все утверждения выражены в формализме дескрипционной логики и записаны на языке OWL. Выполняется это трудоемкое описание автоматически при использовании редакторов. Язык OWL может

быть однозначно прочитан и понят в любой точке всемирной сети и при использовании соответствующих программных средств – парсеров, информация об онтологии будет представлена в понятном для пользователя графическом виде (в виде онтографа или дерева концептов). Дескрипционная логика позволяет осуществлять вывод не заложенных в явном виде знаний, которые также предоставляются пользователю. Онтология, записанная на языке OWL, понята программным агентам и может быть использована ими для обмена знаниями.

Чтобы создать онтологию, необходимо провести анализ предметной области, выявить основные сущности (концепты), построить их иерархию и определить бинарные связи между ними, выработать терминологию, общую для всех пользователей, затем описать свойства классов (концептов) предметной области в виде аксиом тождественности или подчинения.

При описании классов через свойства элементов используются логические формулы (аксиомы). В состав формул входят имена классов, бинарные отношения и логические операции. Любая формула рассматривается как анонимный класс с указанными в формуле свойствами. Именованный класс может быть определен как подкласс или как эквивалент анонимного класса.

Одной из задач данного пособия было показать, как правильно формулировать ограничения на свойства класса в виде логических формул (аксиом).

В пособии приведены примеры различного использования онтологий и дано множество ссылок на источники, где можно найти более подробное изложение соответствующего материала. Использование онтологий в качестве моделей представления знаний – это сравнительно молодое направление в области искусственного интеллекта и большинство публикаций носит исследовательский характер.

Автор надеется, что данное пособие поможет читателю лучше понять смысл и назначение онтологий.

## **Библиографический список**

1. Агеев М.С. Извлечение значимой информации из web-страниц для индексирования / М.С. Агеев, И.В. Вершинников, Б.В. Добров // Электронные библиотеки: перспективные методы и тех-нологии, электронные коллекции: сб. науч. тр. – Ярославль, Россия, 2005.
2. Беляев И.П. Системный анализ для разработки и внедрения информационных технологий / И.П. Беляев, В.М.Капустян.– М.: МГСУ, 2007.
3. Бернерс-Ли, Т. Семантическая Сеть / Т. Бернерс-Ли, О. Лас-сила, Дж. Хендлер // Scientific American. – May. – 2001.
4. Благодаров А.В. Автоматизация проектирования баз данных в среде Sybase PowerDesigner / А.В. Благодаров, Р.В. Тишкун // методические указания к лабораторным работам / Рязан. гос. радиотехн. ун-т; сост.: Рязань. – 2010. – 40 с.
5. Боровикова О.И. Организация порталов знаний на основе онтологий/ О.И.Боровикова, Ю.А. Загорулько // Компьютерная лингвистика и интеллектуальные технологии: сб. науч. тр. – Москва: Наука, – 2002. – Т.2, – С.76-82.
6. Булгаков С.В. Использование онтологий для построения инновационных цепочек в системе поддержки инновационной деятельности в регионе / Булгаков С.В., Загорулько Ю.А. / Проблемы управления и моделирования в сложных системах: сб. науч. тр. – Самара: Самарский Научный Центр РАН. – 2004. – С. 328-333.
7. Гаврилова, Т.А. Базы знаний интеллектуальных систем / Т.А.Гаврилова, В.М. Хорошевский. – СПб: Питер, 2000. – 384 с.
8. Гаврилова, Т.А. Онтологический подход к управлению знаниями при разработке корпоративных систем автоматизации / Т.А. Гаврилова // Новости искусственного интеллекта. – 2003. – № 2. – С. 24-30.
9. Гладун А. Веб-сервисы как основа деловых отношений / А. Гладун // Телеком. Коммуникации и сети. – 2008. – №3. – 56 с.
10. Гладун А.Я. Онтологии в корпоративных системах / А.Я. Гладун, Ю.В. Рогушина // Корпоративные системы. – 2006. – №1
11. Джонс С. На пути к приемлемому определению сервиса / С.Джонс // Открытые системы. – 2005. – №11.

12. Загорулько Ю.А. Организация содержательного доступа к гуманитарным информационным ресурсам на основе онтологий / Загорулько Ю.А., Боровикова О.И., Загорулько Г.Б. // Электронные библиотеки: перспективные методы и технологии, электронные коллекции: сб. науч. тр. – Переславль-Залесский. – Россия.- 2007.
13. Загорулько, Ю.А. Подход к интеллектуализации документооборота / Ю.А. Загорулько [и др.] // Информационные технологии. – 2004. – № 11. – с. 2-11.
14. Каширин Д.И. Структуризация и унификация онтологических описаний на языке OWL в задачах информационного поиска. / А.Н. Пылькин // Проблемы полиграфии и издательского дела. – 2008. – №4. – с. 45-57.
15. Кудрявцев Д. Технологии применения онтологий. / Д. Кудрявцев// Эталонные модели организации деятельности в государственном секторе: НИР/ 2006 г.
16. Лапшин, В. А. Онтологии в компьютерных системах. / В.А. Лапшин// RSDN Magazine. – 4. – 2009.
17. Левшин, Д. Web, часть третья [Текст] / Д. Левшин // Открытые системы. – 2009. – №3.
18. Мирошников В.В. Онтологическая модель системы управления знаниями в области качества / В.В. Мирошников, Д.И. Булатицкий // Вестник Брянского государственного технического университета. – 2009. – № 4 (24).
19. Мирошников В.В. Система управления знаниями в области качества / В.В. Мирошников, Д.И. Булатицкий // Информационные технологии. – 2006. – №7. – С. 16-22.
20. Рассел С., Норвиг П. Искусственный интеллект: современный подход. – М.: ИД «Вильямс», 2006. – 1408 с.
21. Рубашкин В.Ш. Методология наполнения онтологий – практика без теории./ В.Ш. Рубашкин, Л.М. Пивоварова // Онтологическое моделирование / М: ИПИ РАН, 2011.
22. Тузовский А.Ф. Построение модели знаний организации с использованием системы онтологий / А.Ф. Тузовский, С.В. Козлов // Компьютерная лингвистика и интеллектуальные технологии: тр. междунар. конф. «Диалог-2006/ М, 2006. – С. 508-512.
23. Хорошевский, В.Ф. Онтологические модели и Semantic Web: откуда и куда мы идем? / В.Ф Хорошевский // сб. трудов симпозиума «Онтологическое моделирование / М., ИПИ РАН. 2005 г.

- 
24. Цуканова Н.И. Онтология учебно-методического комплекса. / Цуканова Н.И. Страхова З.В. // Вестник Рязанского государственного радиотехнического университета. – №1. – 2013. – 5 с.
25. Цуканова Н.И. Разработка онтологии документальной информационной системы «Учебно-методические комплексы кафедры». / Н.И. Цуканова, З.В. Страхова //Образование в современной России: монография / Москва: Приволжский Дом знаний; МИЭМП, – 2012. – №2. – 120 с.
26. Цуканова Н.И. Разработка онтологии предметной области с использованием редактора Protege 4.1 / Н.И. Цуканова // Методические указания к лабораторным работам / Рязан. гос. радиотехн. ун-т. – Рязань. – 2012. – 52 с.
27. Цуканова Н.И. Теория и практика логического программирования на языке Visual Prolog 7. / Н.И.Цуканова, Т.А. Дмитриева // Учебное пособие для вузов. – М.: Горячая линия – Телеком, – 2011. – 232 с.
28. Шведин Б.Я. Онтология предприятия: экспириентологический подход: Технология построения онтологической модели предприятия. – 2010. – 240 с.
29. A Practical Guide To Building OWL Ontologies Using Prot'eg'e 4 and CO-ODE Tools Edition 1.3.: – The University Of Manchester Copyrightc The University Of Manchester. – 2011.
30. Baader F Patel-Schneider The Description Logic Handbook / F. Baader, D. Calvanese// Theory, Implementation, and Applications. – Cambridge University Press, 2003. – ISBN 0-521-78176-0.
31. Baader F. The Description Logic Handbook. – New York: Cambridge University Press, 2003. – ISBN ISBN 0-521-78176-0.
32. Bacher R., Leal D., Schroder A. ScadaOnWeb – Modelling and Web-Exchange of Process and Engineering Information // Proc. International Semantic Web Conference. – Cagliari. – Italy. – 2002.
33. Description Logic Handbook, 2003, Basic Description Logics, pp. 43-95.
34. Glimm B.; Horrocks I.; Lutz C.; Sattler U. Conjunctive query answering for the description logic SHIQ. In Proc. of the 20th Int. Joint Conf. on Artificial Intelligence. – IJCAI. – 2007. – 151-198.
34. Grädel E.; Otto M.; Rosen E. Two variable logic with counting is decidable. In Proc. of the 12th IEEE Symp. on Logic in Computer Science. – LICS. – 1997. – 306-317.

35. Gruber,T. Towards Principles for the Design of Ontologies Used for Knowledge Sharing // International Workshop on Formal Ontology. – 1993. – Italy.
36. Guarino N. Ontologies and Knowledge Bases. Towards a Terminological Clarification/ N. Guarino, P.Giaretti // Towards Very Large Knowledge Bases.1995-N.J.I.Mars (ed.) IOS Press, Amsterdam.
37. Horrocks I.; Sattler U.; Tobies S. Practical reasoning for expressive Description Logics In Proc. of the 6th Int. Conference on Logic for Programming and Automated Reasoning. – LPAR. – 1999. – 161-180.
38. Lutz C.; Sattler U.; Wolter F. Modal logics and the two-variable fragment. In Annual Conference of the European Association for Computer Science. – Logic. – 2001.
39. Matthew Horridge et al. 2011. A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools, Edition 1.3, published by the University of Manchester, 24 Mar 2011, 108 pp.
40. OWL Web Ontology Language guide. W3C working draft. W3 Consortium: <http://www.w3.org/TR/2003/WD-owl-guide-20030331/>.
41. Schild, K. A correspondence theory for terminological logics: Preliminary report. In Proc. of the 12th Int. Joint Conf. on Artificial Intelligence / IJCAI'91: 1991. – P. 466-471.
42. Schmidt-Schauß M.; Smolka G. Attributive concept descriptions with complements. Artificial Intelligence 48.1991. – P. 1-26.
43. Shankaranarayanan, G.The metadata enigma.-Comm. ACM. – 49(2). – 2006. – P. 88-94.
44. Tessaris, S. Questions and answers: Reasoning and querying in Description Logic (PhD Thesis). – University of Manchester. – 2001.
45. SPARQL Query Language for RDF: <http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/>.
46. Berners-Lee T. Isn't it semantic? (Interview with BCS managing editor Brian Runciman): <http://www.bcs.org/index.php?show=ConWebDoc.3337> – March 2006.
47. Easy RDF and SPARQL for LAMP systems: <http://arc.semsol.org/home>.
48. Hendler J. Current Status and Future Promise of the Semantic Web: <http://www.cs.rpi.edu/academics/courses/fall07/semantic/<http://www.cs.rpi.edu/academics/courses/fall07/semantic/>>Semantics2006-keynote.ppt.

- 
49. OWL Web Ontology Language Guide : <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>
50. W3C Semantic Web FAQ: <http://www.w3.org/2001/sw/SW-FAQ>, [http://apcp.apmath.spbu.ru/ru/staff/tuzov/onapr.html/](http://apcp.apmath.spbu.ru/ru/staff/tuzov/onapr.html).
51. Web-сервисы для новичков: <http://www.ibm.com/developerworks/ru/webservices/newto/websvc.html>.
52. Абрамов А.В. Онтология как метод описания предметных областей. / А.В. Абрамов // Открытые системы – 2005, <http://masters.donntu.edu.ua/2010/fknt/bolotova/library/tez6.htm>
53. Андрей Колесов. Web-сервисы спасут компьютерный мир?: <http://www.visual.2000.ru/kolesov/byte/2001/10816web.htm>  
<http://www.visual.2000.ru/kolesov/byte/2001/10816web.htm>
54. Борис Конев. Онтология и представление знаний: <http://www.lektorium.tv/lecture/?id=13064>
55. Боровикова О.И, Загорулько Ю.А. «Организация порталов знаний на основе онтологий»: <http://hr-portal.ru/article/organizaciya-portalov-znaniy-na-osnove-ontologiy>
56. Веб 2.0: [http://ru.wikipedia.org/wiki/Веб\\_2.0](http://ru.wikipedia.org/wiki/Веб_2.0)
57. Вадим Вирин. Нужен ли Российской Федерации единый классификатор продукции? Computerworld Россия № 9(410) 09/03/2004г.[http://www.intertech.ru/about/comppress.asp?filename=compres\\_43](http://www.intertech.ru/about/comppress.asp?filename=compres_43).
58. Даниил Кальченко «Интеллектуальные агенты семантического Web'a» : <http://www.compress.ru/Archive/CP/2004/10/48/>.
59. Дескрипционная логика (лекции) с.н.с. Евгений Золин: <http://lpcs.math.msu.su/~zolin/dl/>.
60. Ивлев А.А., Артеменко В.Б. Онтология военных технологий: <http://www.milresource.ru/Ontol.html>.
61. Навигатор по вычислительной сложности дескрипционных логик: <http://www.cs.manchester.ac.uk/~ezolin/dl/>.
62. Новые возможности языка OWL 2.0: [http://shcherbak.net/translations/rus.owl2primer\\_shcherbak.net.html](http://shcherbak.net/translations/rus.owl2primer_shcherbak.net.html).
63. НОУ ИНТУИТ. Онтологии и тезаурусы: модели, инструменты, приложения: <http://www.intuit.ru/studies/courses/1078/270/lecture/3672?page=2>.
64. Официальный сайт сообщества исследователей дескрипционных логик: <http://dl.kr.org/>.

65. Гусак П. XML Web Services сервисы: <http://www.itc.ua/node/9484>.
66. Сайт разработчиков языка OWL 1.1: <http://www.w3.org/TR/2004/REC-owl-guide-20040210/>
67. Технология создания web-проекта: <http://info-pages.com.ua/e/43>.
68. W3C Semantic Web FAQ [Электронный ресурс] / W3C. – Режим доступа: www/ URL: <http://www.w3.org/2001/sw/SW-FAQ<http://apcp.apmath.spu.ru/ru/staff/tuzov/onapr.html/>>.
69. <http://www.iacp.dvo.ru/RUS/>.

## Приложение П1

### ПРОГРАММА «СЕМЬЯ» В ФОРМАТЕ, ОСНОВАННОМ НА МАНЧЕСТЕРСКОМ СИНТАКСИСЕ

Таблица П1. Манчестерский синтаксис.  
Ключевые слова и их смысл

Элемент OWL	Пояснение
<b>Prefix:</b>	Описание краткого наименования – префикса
<b>Ontology:</b>	Задание адреса онтологии
<b>Annotations:</b>	Аннотация, т.е. пояснение к программе
<b>AnnotationProperty:</b>	Вид (Тип) аннотации
<b>Datatype:</b>	Тип данных
<b>ObjectProperty:</b>	Свойство объекта (бинарное отношение)
<b>SubPropertyOf:</b>	Подсвойство – описывает подчинение одного отношения другому
<b>Characteristics:</b>	Характеристики отношения
<b>Asymmetric,</b>	Асимметричное отношение
<b>Irreflexive,</b>	Нерефлексивное отношение
<b>Functional</b>	Функциональное отношение
<b>Transitive</b>	Транзитивное отношение
<b>Symmetric</b>	Симметричное отношение
<b>Domain:</b>	Область определения
<b>Range:</b>	Область значений
<b>InverseOf:</b>	Обратное (инверсное) отношение
<b>SubPropertyChain:</b>	Цепочка (композиция) отношений
<b>DisjointWith:</b>	Не пересекается с ...
<b>inverse</b>	Обратное (инверсное)
<b>Class:</b>	Класс
<b>EquivalentTo:</b>	Эквивалентный ....
<b>and</b>	Конъюнкция (операция «И»)
<b>or</b>	Дизъюнкция (операция «ИЛИ»)
<b>value</b>	Значение
<b>not</b>	Операция «НЕ» (Отрицание, дополнение)
<b>some</b>	Существует хотя бы один элемент ... (Квантор существования $\exists$ )
<b>only</b>	Связь только с элементами ... (Квантор всеобщности $\forall$ )

## Продолжение таблицы III

Элемент OWL	Пояснение
<b>Individual:</b>	Индивидуальность
<b>Types:</b>	Типы
<b>Facts:</b>	Факты
<b>SameAs:</b>	Подобные объекты ....
<b>min</b>	Минимальное значение кардинальности, кардинальность >= min
<b>exactly</b>	Точное значение кардинальности, кардинальность = <число>
<b>max</b>	Максимальное значение кардинальности, кардинальность <= max
<b>DifferentIndividuals:</b>	Различные индивидуальности

**Программа .**

Prefix: :  
<http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-88#>  
Prefix: owl: <http://www.w3.org/2002/07/owl#>  
Prefix: rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>  
Prefix: xml: <http://www.w3.org/XML/1998/namespace>  
Prefix: xsd: <http://www.w3.org/2001/XMLSchema#>  
Prefix: rdfs: <http://www.w3.org/2000/01/rdf-schema#>  
Prefix: СемьяПример-ontology-88:  
<http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-88#>  
Ontology:  
<http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-88>

**Annotations:**

rdfs:comment "Онтология семьи. В ней общие знания о семье применяются к двум семьям, для которых заданы индивидуальности и их отношения друг к другу."  
^^xsd:string,  
owl:backwardCompatibleWith  
"http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-88"  
AnnotationProperty: rdfs:seeAlso  
AnnotationProperty: owl:backwardCompatibleWith  
AnnotationProperty: rdfs:label

AnnotationProperty: rdfs:comment  
  
Datatype: rdf:PlainLiteral  
Datatype: xsd:string  
Datatype: xsd:integer  
ObjectProperty: имеет\_жену  
    SubPropertyOf:  
        имеет\_супруга  
    Characteristics:  
        Asymmetric,  
        Irreflexive,  
        Functional  
    Domain:  
        Человек  
    Range:  
        Человек  
    InverseOf:  
        имеет\_мужа  
ObjectProperty: имеет\_предка  
    Characteristics:  
        Transitive  
ObjectProperty: имеет\_внуков  
    SubPropertyChain:  
        имеет\_ребенка о имеет\_ребенка  
ObjectProperty: имеет\_дочь  
    SubPropertyOf:  
        имеет\_ребенка  
    DisjointWith:  
        имеет\_сына  
    Domain:  
        Человек  
    Range:  
        Человек  
ObjectProperty: имеет\_сына  
    SubPropertyOf:  
        имеет\_ребенка  
    DisjointWith:  
        имеет\_дочь  
    Domain:  
        Человек  
    Range:  
        Человек  
ObjectProperty: имеет\_супруга

Characteristics:  
    Irreflexive,  
    Symmetric

ObjectProperty: имеет\_ребенка  
    SubPropertyOf:  
        inverse (имеет\_предка)

Characteristics:  
    Irreflexive

Domain:  
    Человек

Range:  
    Человек

ObjectProperty: любит  
    Characteristics:  
        Asymmetric

ObjectProperty: имеет\_мужа  
    SubPropertyOf:  
        имеет\_супруга

Characteristics:  
    Asymmetric,  
    Irreflexive,  
    InverseFunctional

Domain:  
    Человек

Range:  
    Человек

InverseOf:  
    имеет\_жену

DataProperty: имеет\_возраст  
    Characteristics:  
        Functional

Domain:  
    Человек

Range:  
    xsd:integer

Class: owl:Thing

Class: ДедИлиБабушка  
    EquivalentTo:  
        Человек  
            and (имеет\_внуков some Человек)

Class: Потомок  
    EquivalentTo:  
        Человек

```
        and (имеет_предка value Наташа)
SubClassOf:
    Человек
Class: Подросток
    EquivalentTo:
        Человек
and (имеет_возраст some xsd:integer[>= 13 , < 21])
    SubClassOf:
        Человек
Class: ЗаконныйБрак
    SubClassOf:
        Брак
Class: Ребенок_Дитя
    EquivalentTo:
        Человек
and (not (имеет_возраст some xsd:integer[>= 21]))
    SubClassOf:
        Человек
Class: Родитель
    EquivalentTo:
        Человек
        and (имеет_ребенка some Человек)
Class: Взрослый
    EquivalentTo:
        Человек
        and (имеет_возраст some xsd:integer[>= 21])
    SubClassOf:
        Человек
Class: Мужчина
    SubClassOf:
        Человек
    DisjointWith:
        Женщина
Class: Нарциссит
    Annotations:
        rdfs:label "Самовлюбленный",
        rdfs:seeAlso
<
```

Class: ГражданскийБрак  
SubClassOf:  
    Брак

Class: Брак

Class: Человек

Class: Женщина  
SubClassOf:  
    Человек

DisjointWith:  
    Мужчина

Individual: Нина  
Types:  
    Женщина

Facts:  
    имеет\_возраст 13

Individual: Володя  
Types:  
    Мужчина

Facts:  
    любит Володя,  
    имеет\_возраст 5

Individual: Наташа  
Types:  
    Женщина,  
    имеет\_возраст some {55 , 65}

Facts:  
    имеет\_ребенка Федор,  
    имеет\_ребенка Дуся

Individual: Иван  
Annotations:  
    rdfs:label "Ваня"

Types:  
    Мужчина

Facts:  
    любит Евдокия,  
    имеет\_жену Евдокия,  
    имеет\_сына Володя,  
    имеет\_дочь Нина,  
    имеет\_возраст 50

Individual: Дуся  
Annotations:  
    rdfs:label "Евдокия"^^xsd:string,

rdfs:comment "Дуся и Евдокия - одно и то же лицо. Поэтому у Дуси есть дети: Володя и Нина"^^xsd:string,  
rdfs:seeAlso  
<<http://www.semanticweb.org/ontologies/2013/9/СемьяПример-ontology-88#Евдокия>>

Types:  
not (имеет\_возраст value 37),  
Женщина

SameAs:  
Евдокия

Individual: Сергей

Types:  
Мужчина,  
имеет\_ребенка exactly 2 owl:Thing,  
имеет\_ребенка min 1 Мужчина

Facts:  
имеет\_ребенка Дуся,  
имеет\_ребенка Федор,  
имеет\_жену Наташа,  
имеет\_возраст 77

Individual: Федор

Types:  
Мужчина,  
имеет\_возраст some xsd:integer[>= 15 , < 21]

Individual: Евдокия

Types:  
Женщина

Facts:  
имеет\_сына Володя,  
имеет\_дочь Нина,  
имеет\_возраст 40

SameAs:  
Дуся

DifferentIndividuals: Дуся, Наташа, Сергей, Федор

## Приложение П2

### ПРОГРАММА «СЕМЬЯ» В ФОРМАТЕ, ОСНОВАННОМ НА ФУНКЦИОНАЛЬНОМ СИНТАКСИСЕ

Таблица П2. Ключевые слова функционального синтаксиса

Элемент OWL	Пояснение
<b>Prefix:</b>	Описание краткого наименования - префикса
<b>Ontology:</b>	Задание адреса онтологии
<b>Annotation:</b>	Аннотация, т.е. пояснение к программе
<b>Declaration</b>	Объявление (описание)
<b>Class:</b>	Класс
<b>SubClassOf</b>	Подкласс класса
<b>EquivalentClasses</b>	Эквивалентные классы ....
<b>DisjointUnion</b>	Объединение непересекающихся множеств
<b>ObjectIntersectionOf</b>	Пересечение
<b>DataSomeValuesFrom</b>	Существует хотя бы одно значение данных из заданного диапазона
<b>DatatypeRestriction</b>	Ограничение на диапазон возможных значений данного
<b>minInclusive</b>	$\geq <\text{число}>$ Минимальное значение
<b>maxExclusive</b>	$\leq <\text{число}>$ Максимальное значение
<b>ObjectSomeValuesFrom</b>	Существует хотя бы один объект из заданного класса ...
<b>AnnotationAssertion</b>	Утверждение - аннотация
<b>ObjectHasSelf</b>	Объект, являющийся в отношении последователем самого себя
<b>DisjointClasses</b>	Непересекающиеся классы
<b>ObjectInverseOf</b>	Обратно (инверсно) ....
<b>NamedIndividual</b>	Имя индивидуальности
<b>ClassAssertion</b>	Элемент принадлежит классу
<b>ObjectPropertyAssertion</b>	Принадлежность элементов отношению
<b>DataPropertyAssertion</b>	Описание связи объекта с данными
<b>ObjectComplementOf</b>	Дополнение (Отрицание)
<b>SameIndividual</b>	Подобные индивидуальности
<b>DataOneOf</b>	Одно из значений списка

*Продолжение таблицы П2*

<b>Элемент OWL</b>	<b>Пояснение</b>
<b>ObjectProperty:</b>	Свойство объекта (бинарное отношение)
<b>SubObjectPropertyOf</b>	Подсвойство – описывает подчинение одного отношения другому
<b>ObjectPropertyDomain</b>	Область определения отношения
<b>ObjectPropertyRange</b>	Область значений отношения
<b>DisjointObjectProperties</b>	Непересекающиеся отношения
<b>InverseObjectProperties</b>	Обратное (инверсное) отношение
<b>FunctionalObjectProperty</b>	Функциональное отношение
<b>AsymmetricObjectProperty</b>	Асимметричное отношение
<b>IrreflexiveObjectProperty</b>	Нерефлексивное отношение
<b>TransitiveObjectProperty</b>	Транзитивное отношение
<b>SymmetricObjectProperty</b>	Симметричное отношение
<b>InverseFunctionalObjectProperty</b>	Инверсное функциональное отношение
<b>ObjectPropertyChain</b>	Цепочка (композиция) отношений

**Программа.**

```

Prefix(:=<http://www.semanticweb.org/ontologies/2013/
9/СемьяПример-ontology-88#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(rdf:=<http://www.w3.org/1999/02/22-rdf-syntax-
ns#>)
Prefix(xml:=<http://www.w3.org/XML/1998/namespace>)
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(rdfs:=<http://www.w3.org/2000/01/rdf-schema#>)
Prefix(СемьяПример-ontology-
88:=<http://www.semanticweb.org/ontologies/2013/9/Сем
ьяПример-ontology-88#>)


```

Ontology(<http://www.semanticweb.org/ontologies/2013/
9/СемьяПример-ontology-88>

Annotation(rdfs:comment "Онтология семьи. В ней общие знания о семье применяются к двум семьям, для которых заданы индивидуальности и их отношения друг к другу."^^xsd:string)

```
Annotation(owl:backwardCompatibleWith
"http://www.semanticweb.org/ontologies/2013/9/СемьяПр
имер-ontology-88")

Declaration(Class(:Брак))
DisjointUnion(:Брак :ЗаконныйБрак :ГражданскийБрак)
Declaration(Class(:Взрослый))
EquivalentClasses(:Взрослый
ObjectIntersectionOf(DataSomeValuesFrom(:имеет_возрас
т DatatypeRestriction(xsd:integer xsd:minInclusive
"21"^^xsd:integer)) :Человек))
SubClassOf(:Взрослый :Человек)
Declaration(Class(:ГражданскийБрак))
SubClassOf(:ГражданскийБрак :Брак)
Declaration(Class(:ДедИлиБабушка))
EquivalentClasses(:ДедИлиБабушка
ObjectIntersectionOf(ObjectSomeValuesFrom(:имеет_вну
ков :Человек) :Человек))
Declaration(Class(:Женщина))
SubClassOf(:Женщина :Человек)
DisjointClasses(:Женщина :Мужчина)
Declaration(Class(:ЗаконныйБрак))
SubClassOf(:ЗаконныйБрак :Брак)
Declaration(Class(:Мужчина))
SubClassOf(:Мужчина :Человек)
DisjointClasses(:Мужчина :Женщина)
Declaration(Class(:Нарциссит))
AnnotationAssertion(rdfs:comment :Нарциссит "Человек,
который любит самого себя")
AnnotationAssertion(rdfs:label :Нарциссит
"Самовлюбленный")
AnnotationAssertion(rdfs:seeAlso :Нарциссит :Володя)
EquivalentClasses(:Нарциссит
ObjectIntersectionOf(ObjectHasSelf(:любит) :Человек))
Declaration(Class(:Подросток))
EquivalentClasses(:Подросток
ObjectIntersectionOf(DataSomeValuesFrom(:имеет_возрас
т DatatypeRestriction(xsd:integer xsd:maxExclusive
"21"^^xsd:integer xsd:minInclusive
"13"^^xsd:integer)) :Человек))
```

```
SubClassOf(:Подросток :Человек)
Declaration(Class(:Потомок))
EquivalentClasses(:Потомок
ObjectIntersectionOf(ObjectHasValue(:имеет_предка
:Наташа) :Человек))
SubClassOf(:Потомок :Человек)
Declaration(Class(:Ребенок_дитя))
EquivalentClasses(:Ребенок_дитя
ObjectIntersectionOf(ObjectComplementOf(DataSomeValuesFrom(:имеет_возраст DatatypeRestriction(xsd:integer
xsd:minInclusive "21"^^xsd:integer))) :Человек))
SubClassOf(:Ребенок_дитя :Человек)
Declaration(Class(:Родитель))
EquivalentClasses(:Родитель
ObjectIntersectionOf(ObjectSomeValuesFrom(:имеет_ребенка :Человек) :Человек))
Declaration(Class(:Человек))
Declaration(ObjectProperty(:имеет_внуков))
Declaration(ObjectProperty(:имеет_дочь))
SubObjectPropertyOf(:имеет_дочь :имеет_ребенка)
ObjectPropertyDomain(:имеет_дочь :Человек)
ObjectPropertyRange(:имеет_дочь :Человек)
DisjointObjectProperties(:имеет_дочь :имеет_сына)
Declaration(ObjectProperty(:имеет_жену))
SubObjectPropertyOf(:имеет_жену :имеет_супруга)
InverseObjectProperties(:имеет_жену :имеет_мужа)
FunctionalObjectProperty(:имеет_жену)
AsymmetricObjectProperty(:имеет_жену)
IrreflexiveObjectProperty(:имеет_жену)
ObjectPropertyDomain(:имеет_жену :Человек)
ObjectPropertyRange(:имеет_жену :Человек)
Declaration(ObjectProperty(:имеет_мужа))
SubObjectPropertyOf(:имеет_мужа :имеет_супруга)
InverseObjectProperties(:имеет_жену :имеет_мужа)
InverseFunctionalObjectProperty(:имеет_мужа)
AsymmetricObjectProperty(:имеет_мужа)
IrreflexiveObjectProperty(:имеет_мужа)
ObjectPropertyDomain(:имеет_мужа :Человек)
ObjectPropertyRange(:имеет_мужа :Человек)
Declaration(ObjectProperty(:имеет_предка))
```

```
TransitiveObjectProperty(:имеет_предка)
Declaration(ObjectProperty(:имеет_ребенка))
SubObjectPropertyOf(:имеет_ребенка
ObjectInverseOf(:имеет_предка))
IrreflexiveObjectProperty(:имеет_ребенка)
ObjectPropertyDomain(:имеет_ребенка :Человек)
ObjectPropertyRange(:имеет_ребенка :Человек)
Declaration(ObjectProperty(:имеет_супруга))
SymmetricObjectProperty(:имеет_супруга)
IrreflexiveObjectProperty(:имеет_супруга)
Declaration(ObjectProperty(:имеет_сына))
SubObjectPropertyOf(:имеет_сына :имеет_ребенка)
ObjectPropertyDomain(:имеет_сына :Человек)
ObjectPropertyRange(:имеет_сына :Человек)
DisjointObjectProperties(:имеет_сына :имеет_дочь)
Declaration(ObjectProperty(:любит))
AsymmetricObjectProperty(:любит)
Declaration(DataProperty(:имеет_возраст))
FunctionalDataProperty(:имеет_возраст)
DataPropertyDomain(:имеет_возраст :Человек)
DataPropertyRange(:имеет_возраст xsd:integer)
Declaration(NamedIndividual(:Володя))
ClassAssertion(:Мужчина :Володя)
ObjectPropertyAssertion(:любит :Володя :Володя)
DataPropertyAssertion(:имеет_возраст :Володя
"5"^^xsd:integer)
Declaration(NamedIndividual(:Дуся))
AnnotationAssertion(rdfs:comment :Дуся "Дуся и
Евдокия - одно и то же лицо. Поэтому у Дуси есть
дети: Володя и Нина"^^xsd:string)
AnnotationAssertion(rdfs:seeAlso :Дуся :Евдокия)
AnnotationAssertion(rdfs:label :Дуся
"Евдокия"^^xsd:string)
ClassAssertion(:Женщина :Дуся)
ClassAssertion(ObjectComplementOf(DataHasValue(:имеет_
_возраст "37"^^xsd:integer)) :Дуся)
SameIndividual(:Дуся :Евдокия)
Declaration(NamedIndividual(:Евдокия))
ClassAssertion(:Женщина :Евдокия)
SameIndividual(:Евдокия :Дуся)
```

```
ObjectPropertyAssertion(:имеет_дочь :Евдокия :Нина)
ObjectPropertyAssertion(:имеет_сына :Евдокия :Володя)
DataPropertyAssertion(:имеет_возраст :Евдокия
"40"^^xsd:integer)
Declaration(NamedIndividual(:Иван))
AnnotationAssertion(rdfs:label :Иван "Ваня")
ClassAssertion(:Мужчина :Иван)
ObjectPropertyAssertion(:имеет_дочь :Иван :Нина)
ObjectPropertyAssertion(:имеет_жену :Иван :Евдокия)
ObjectPropertyAssertion(:имеет_сына :Иван :Володя)
ObjectPropertyAssertion(:любит :Иван :Евдокия)
DataPropertyAssertion(:имеет_возраст :Иван
"50"^^xsd:integer)
Declaration(NamedIndividual(:Наташа))
ClassAssertion(:Женщина :Наташа)
ClassAssertion(DataSomeValuesFrom(:имеет_возраст
DataOneOf("65"^^xsd:integer "55"^^xsd:integer))
:Наташа)
ObjectPropertyAssertion(:имеет_ребенка :Наташа :Дуся)
ObjectPropertyAssertion(:имеет_ребенка :Наташа
:Федор)
Declaration(NamedIndividual(:Нина))
ClassAssertion(:Женщина :Нина)
DataPropertyAssertion(:имеет_возраст :Нина
"13"^^xsd:integer)
Declaration(NamedIndividual(:Сергей))
ClassAssertion(:Мужчина :Сергей)
ClassAssertion(ObjectMinCardinality(1 :имеет_ребенка
:Мужчина) :Сергей)
ClassAssertion(ObjectExactCardinality(2
:имеет_ребенка) :Сергей)
ObjectPropertyAssertion(:имеет_жену :Сергей :Наташа)
ObjectPropertyAssertion(:имеет_ребенка :Сергей
:Федор)
ObjectPropertyAssertion(:имеет_ребенка :Сергей :Дуся)
DataPropertyAssertion(:имеет_возраст :Сергей
"77"^^xsd:integer)
Declaration(NamedIndividual(:Федор))
ClassAssertion(:Мужчина :Федор)
```

```
ClassAssertion(DataSomeValuesFrom(:имеет_возраст
DatatypeRestriction(xsd:integer xsd:minInclusive
"15"^^xsd:integer xsd:maxExclusive
"21"^^xsd:integer)) :Федор)
DifferentIndividuals(:Дуся :Наташа :Сергей :Федор)
SubObjectPropertyOf(ObjectPropertyChain(:имеет_ребенк
а :имеет_ребенка) :имеет_внуков)
)
```