

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
Пензенский государственный университет

И. Ю. Балашова, Е.Н. Прошкина

**Онтологический инжиниринг в
проектировании интеллектуальных
информационных систем**

Учебное пособие

Пенза 2018

Рецензенты:

Бутаев М.М., д.т.н., профессор, ученый секретарь АО «НПП «Рубин»

Пашенко Д.В., д.т.н., профессор, заведующий кафедрой «Вычислительная техника» ПГУ

В учебном пособии систематически изложены теоретические основы онтологического инжиниринга в области информатики. Рассмотрены методы и принципы онтологического инжиниринга применительно к построению и проектированию интеллектуальных информационных систем. Описана методика проектирования информационных систем и компьютерных онтологий произвольных предметных областей. Отдельное внимание удалено процессу создания онтологий для заданной предметной области с использованием редактора онтологий Protégé.

Учебное пособие подготовлено на кафедре «Математическое обеспечение и применение ЭВМ» и предназначено для магистров, обучающихся по направлению 09.04.02 «Информационные системы и технологии».

ГЛАВА 1. ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ИНТЕЛЛЕКТУАЛЬНЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ

1.1. Данные и знания

Интеллектуальные информационные системы (ИИС) или системы, основанные на знаниях, представляют комплекс программных, лингвистических и логико-математических средств для реализации основной задачи – осуществления поддержки деятельности человека и поиска информации в режиме продвинутого диалога на естественном языке. Системы, основанные на знаниях, применяются во многих предметных областях для решения сложных слабоструктурированных проблем. При изучении подобных систем возникает вопрос – что же такое знания и чем они отличаются от данных.

Данные – это поддающееся многократной интерпретации представление информации в формализованном виде, пригодном для хранения, передачи и обработки. Данные могут быть цифровыми (факты, результаты измерений), графическими, аудио, видео и т.п. Они могут описываться на различных языках (символьном, математическом, графическом и т.п.). Качественными мерами для данных являются своевременность, соответствие и точность.

Знания являются более сложной категорией информации по сравнению с данными. Под знаниями понимается форма представления информации, которой присущи такие особенности, как:

- Внутренняя интерпретируемость (каждая информационная единица имеет уникальное имя, по которому система находит ее, а также отвечает на запросы, в которых это имя упомянуто).
- Структурированность (включенность одних информационных единиц в состав других).

- Связность (возможность задания временных, причинных, пространственных или иного рода отношений).
- Семантическая метрика (возможность задания отношений, характеризующих ситуационную близость).
- Активность (выполнение программ инициируется текущим состоянием информационной базы).

По своей природе знания можно разделить на декларативные и процедурные. Декларативные знания содержат в себе лишь представление о структуре некоторых понятий. Эти знания приближены к данным. Декларативные знания представляют собой описание фактов и явлений, фиксируют наличие или отсутствие таких фактов, а также включают описание основных связей и закономерностей, в которые эти факты и явления входят. Например: высшее учебное заведение есть совокупность факультетов, а каждый факультет в свою очередь есть совокупность кафедр. Процедурные знания имеют активную природу. Процедурные знания – это описания действий, которые возможны при манипулировании фактами и явлениями для достижения намеченных целей. Процедурные знания определяют представления о средствах и путях получения новых знаний, проверки знаний. Примерами процедурных знаний являются алгоритмы разного рода.

По способу приобретения знания можно разделить на факты и эвристику (правила, которые позволяют сделать выбор при отсутствии точных теоретических обоснований). Первая категория знаний обычно указывает на хорошо известные в данной предметной области обстоятельства. Вторая категория знаний основана на собственном опыте эксперта, работающего в конкретной предметной области, накопленном в результате многолетней практики.

По типу представления знания делятся на факты и правила. Факты — это знания типа «*A* – это *A*», такие знания характерны для баз данных и сетевых моделей. Правила или продукции – это знания типа «ЕСЛИ *A*, ТО *B*». Кроме фактов и правил существуют еще метазнания — знания о знаниях.

Они необходимы для управления базой знаний и для эффективной организации процедур логического вывода.

Состав знаний в ИИС зависит от:

- предметной области;
- назначения и структуры;
- требований и целей пользователей;
- языка общения и способах организации диалога с пользователем.

Интеллектуальные информационные системы базируются на концепции использования базы знаний для генерации алгоритмов решения прикладных задач различных классов в зависимости от конкретных информационных потребностей пользователей. База знаний (БЗ) представляет собой особого рода базу данных, разработанную для оперирования знаниями. В отличие от базы данных, БЗ содержат в себе не только фактическую информацию, но и правила вывода, позволяющие делать автоматические умозаключения об уже имеющихся или вновь вводимых фактах и тем самым производить семантическую (осмысленную) обработку информации.

1.2. Концептуальная парадигма работы со знаниями

Концептуальная парадигма работы со знаниями утвердилась в связи и по мере развития концепции интеллектуальных информационных систем, согласно которой главной информационной единицей компьютерной обработки стало «знание». Согласно данной парадигме работа со знаниями рассматривается как процесс, в котором выделяют следующие подпроцессы:

- извлечение знаний;
- приобретение знаний;
- представление знаний;
- манипулирование знаниями.

В область аспектов работы со знаниями, кроме указанных подпроцессов, включают также методы и средства.

1. *Извлечение знаний* из различных источников включает два основных процесса:

- анализ исходной информации и формализацию качественных знаний;
- интеграцию знаний.

Первый процесс связан с созданием методов, позволяющих переходить от знаний, выраженных, в том числе в естественно-языковой форме, к их аналогам, пригодным для ввода в память знание-ориентированной информационной системы. Второй процесс связан с интеграцией знаний, получаемых от различных источников, в некоторую взаимосвязанную и непротиворечивую систему знаний о предметной области.

2. *Приобретение знаний*. Знаний, содержащихся в источниках информации, отчуждённых от специалиста, как правило, недостаточно. Значительную часть профессионального опыта эти специалисты не могут выразить словесно (профессиональное умение или интуиция). Поэтому для того, чтобы приобрести такие знания, нужны специальные приёмы и методы. Они используются в инструментальных системах по приобретению знаний, создание которых – одна из задач инженерии знаний. Полученные от экспертов знания нужно оценить с точки зрения их соответствия ранее накопленным знаниям и формализовать для ввода в память системы. Кроме того, знания, полученные от различных экспертов, необходимо согласовать между собой, так как нередки случаи, когда эти знания оказывались внешне несовместимыми и даже противоречивыми. Рассматриваемый подпроцесс включает такие процессы как:

- организация работы с экспертами;
- оценка и формализация знаний;
- согласование знаний.

3. Представление знаний. Этот процесс предполагает разработку формальной научной теории, включающей построение *модели знаний, системы представления знаний и базы знаний*.

Представление знаний – это соглашение о том, как и в какой формальной теории описывать реальный мир. В естественных и технических науках принят следующий традиционный способ представления знаний. На естественном языке вводятся основные понятия и отношения между ними. Но при этом используются ранее определённые понятия и отношения, смысл которых уже известен. Далее устанавливается соответствие между характеристиками (чаще всего количественными) понятий и подходящей математической моделью. Основная цель представления знаний – строить математические модели реального мира и его частей. С этой целью созданы и используются в действующих системах различные модели представления знаний. Каждая модель знаний определяет форму представления знаний и является формализмом, призванным отобразить объекты, связи между ними и отношения, иерархию понятий предметной области и изменение отношений между объектами

Выделяют два типа моделей представления знаний: формальные и неформальные. К неформальным моделям относят сетевые, производственные модели, фреймы. В отличие от формальных моделей, в основе которых лежит строгая математическая теория, неформальные модели такой теории не придерживаются. Каждая неформальная модель годится только для конкретной предметной области и поэтому не обладает универсальностью, которая присуща моделям формальным. Логический вывод в формальных системах строг и корректен, поскольку подчинен жестким аксиоматическим правилам. Вывод в неформальных системах во многом определяется самим исследователем, который и отвечает за его корректность. Однако в формальных моделях очень сложно учитывать динамические изменения в предметной области. Поэтому такой способ представления знаний

используются в тех предметных областях, которые хорошо локализуются и мало зависят от внешних факторов.

В инженерии знаний системы представления знаний включают совокупность процедур, необходимых для записи знаний, извлечение их из памяти и поддержки хранения знаний в рабочем состоянии. Системой представления знаний называют совокупность средств, позволяющих:

- описывать знания о предметной области с помощью языка представления знаний;
- организовать хранение знаний в системе (накопление, анализ, структурное обобщение и организация знаний);
- выводить новые знания из имеющихся и объединять их;
- находить требуемые знания;
- обновлять знания;
- осуществлять интерфейс между системой и пользователем.

Системы представления знаний оформляются как базы знаний.

4. *Манипулирование знаниями.* К этому процессу относятся такие процессы как

- пополнение знаний;
- классификация знаний;
- обобщение знаний;
- вывод на знаниях;
- рассуждения с помощью знаний;
- объяснения на знаниях;
- решение прикладных задач предметной области.

Новые знания, поступающие в БЗ, должны вместе с теми сведениями, которые уже были ранее записаны в неё, сформировать расширение поступивших знаний. Знания образуют упорядоченные структуры, что облегчает поиск нужных знаний и поддержание работоспособности БЗ. Для этого используются различные классифицирующие процедуры:

родовидовые, «часть–целое», ситуативные (когда в одно множество объединяются знания, релевантные некоторой типовой ситуации). В процессе классификации часто происходит абстрагирование от отдельных элементов описаний (фрагментов знаний об объектах или явлениях), появляются обобщённые знания, что приводит, в конце концов, к абстрактным знаниям, для которых нет прямого прообраза во внешнем мире.

Вывод на знаниях зависит от модели, которая используется для их представления. Появления в памяти информационной системы новых фактов и сведений может повлиять на смену исходных аксиом в процессе вывода. Еще одна особенность вывода на знаниях – неполнота сведений о предметной области и протекающих в ней процессах, неточность входной информации. А это означает, что выводы в системах, основанных на знаниях, носят не абсолютно достоверный характер, как в традиционных логических системах, а приближенный, правдоподобный характер. Такие выводы требуют развитого аппарата вычисления оценок правдоподобия и методов оперирования ими.

Решающим аспектом разработки интеллектуальной системы, предопределяющим ее возможности, свойства и характеристики, является выбор модели представление знаний. С точки зрения человека, желательно, чтобы описательные возможности используемой модели были как можно выше. С другой стороны, сложное представление знаний требует специальных способов обработки (усложняется механизм вывода), что затрудняет проектирование и реализацию интеллектуальной подсистемы. Кроме того, используемый в интеллектуальной системе формализм представления знаний определяет характер их получения и накопления, в результате которого создается БЗ, ориентированная на определенную структуру представления, а не на сущность самих знаний. Выбор модели, неадекватной типам знаний, приводит к потере многих существенных деталей прикладной задачи и порождает тривиальный интеллект.

1.3. Представление знаний с использованием формальных логических систем

В основе логических моделей лежит формальная система взаимодействия ряда множеств, задаваемая выражением вида $M = M = \langle T, P, A, B \rangle$. Множеством T представлены базовые элементы различной природы, например, слова из некоторого ограниченного словаря и т. п. При этом для множества T существует некоторый способ определения принадлежности или непринадлежности произвольного элемента к этому множеству. Процедура проверки принадлежности $\Pi(T)$ должна за конечное число шагов дать положительный или отрицательный ответ на вопрос, является ли элемент x частью множества T .

Множество P определяет набор правил, при помощи которых из элементов множества T образуются синтаксически правильные совокупности. Например, из слов ограниченного словаря строятся синтаксически правильные фразы. Декларируется существование процедуры $\Pi(P)$, с помощью которой за конечное число шагов можно получить ответ на вопрос, является ли совокупность X синтаксически правильной.

Во множестве синтаксически правильных совокупностей выделяется некоторое подмножество A , элементы которого называются аксиомами. Для этого множества существует своя процедура $\Pi(A)$, с помощью которой для любой синтаксически правильной совокупности можно получить ответ на вопрос о принадлежности ее к множеству A .

Множество B определяет набор правил вывода. Применяя эти правила к элементам подмножества A , можно получать новые синтаксически правильные совокупности, к которым снова можно применять правила из B . Так формируется множество выводимых в данной формальной системе совокупностей. Если имеется процедура $\Pi(B)$, при помощи которой можно определить для любой синтаксически правильной совокупности, является ли она выводимой, то соответствующая формальная система называется

разрешимой. Это показывает, что именно правило вывода является наиболее сложной составляющей формальной системы.

Для знаний, входящих в базу знаний, можно считать, что множество A образуют все информационные единицы, которые введены в базу знаний извне, а при помощи правил вывода из них выводятся новые производные знания. Другими словами, формальная система представляет собой генератор порождения новых знаний, образующих множество выводимых в данной системе знаний. Это свойство логических моделей позволяет хранить в базе лишь те знания, которые образуют множество A , а все остальные знания получать из них по правилам вывода.

Основными преимуществами формальных логических моделей являются наличие формального аппарата вывода новых фактов (знаний) из известных фактов (знаний), возможность контроля целостности. Однако в данной модели знания трудно структурировать и поэтому к предметной области предъявляются высокие требования и ограничения.

1.4. Сетевые модели представления знаний

Сетевые модели формально можно задать в виде

$$H = \langle I, C_1, C_2, \dots, C_n, G \rangle,$$

где I – множество информационных единиц; C_1, C_2, \dots, C_n — множество типов связей между информационными единицами; G задает отображение между информационными единицами, входящими в I , посредством связей из заданного набора их типов.

В зависимости от используемых в сетевых моделях типов связей различают классифицирующие сети, функциональные сети и сценарии.

В классифицирующих сетях используются отношения структуризации. Такие сети позволяют в базах знаний вводить разные иерархические отношения между информационными единицами.

Функциональные сети характеризуются наличием функциональных отношений. Их часто называют вычислительными моделями, т. к. они

позволяют описывать процедуры вычисления одних информационных единиц через другие.

В сценариях используются каузальные отношения, а также отношения типов «средство–результат», «орудие–действие» и т. п.

Если в сетевой модели допускаются связи различного типа, то ее обычно называют семантической сетью. *Семантическая сеть* – информационная модель предметной области, имеющая вид ориентированного графа, вершины которого соответствуют объектам предметной области, а дуги (ребра) задают отношения между ними. Наиболее часто в семантических сетях используются следующие отношения:

- связи типа «часть – целое» («класс – подкласс», «элемент – множество» и т. п.);
- функциональные связи (определяемые обычно глаголами «производит», «влияет»...);
- количественные (больше, меньше, равно...);
- пространственные (далеко от, близко от, за, под, над...);
- временные (раньше, позже, в течение...);
- атрибутивные связи (иметь свойство, иметь значение);
- логические связи (И, ИЛИ, НЕ);
- лингвистические связи и др.

Для всех семантических сетей справедливо разделение по арности и количеству типов отношений.

По количеству типов отношений сети могут быть *однородными* и *неоднородными*. Однородные сети обладают только одним типом отношений. В неоднородных сетях количество типов отношений больше одного. Классические иллюстрации данной модели представления знаний представляют именно такие сети. Неоднородные сети представляют больший интерес для практических целей, но и большую сложность для исследования. Неоднородные сети можно представлять как переплетение древовидных многослойных структур.

По арности выделяют сети с *бинарными* и *n-арными* отношениями. Типичными являются сети с бинарными отношениями (связывающими ровно два понятия). Бинарные отношения очень просты и удобно изображаются на графике в виде стрелки между двух концептов. Кроме того, они играют исключительную роль в математике. На практике, однако, могут понадобиться отношения, связывающие более двух объектов – n-арные. При этом возникает сложность в изображении подобной связи на графике. Концептуальные графы снимают это затруднение, представляя каждое отношение в виде отдельного узла.

Проблема поиска решения в базе знаний типа семантической сети сводится к задаче поиска фрагмента сети, соответствующего некоторой подсети, отражающей поставленный запрос к базе.

Достоинство семантических сетей – наглядность представления знаний, с их помощью удобно представлять причинно-следственные связи между элементами (подсистемами), а также структуру сложных систем. Недостаток таких сетей – сложность вывода, поиска подграфа, соответствующего запросу.

1.5. Продукционные модели представления знаний

В продукционных моделях используются некоторые элементы логических и сетевых моделей. Из логических моделей заимствована идея правил вывода, которые здесь называются продукциями, а из сетевых моделей – описание знаний в виде семантической сети. В результате применения правил вывода к фрагментам сетевого описания происходит трансформация семантической сети за счет смены ее фрагментов, наращивания сети и исключения из нее ненужных фрагментов.

В общем случае продукционную модель можно представить в следующем виде:

$$i = \langle S, P, A \Rightarrow B, Q \rangle,$$

где i – имя продукции;

S – сфера применения продукции;

P – условие применимости ядра продукции;

$A \Rightarrow B$ – ядро продукции, в котором A – условие ядра (или антецентент), B – заключение ядра (или консеквент) , « \Rightarrow » – знак логической секвенции (или следования);

Q – постусловие производственного правила, актуализирующееся при положительной реализации продукции.

В производственных моделях процедурная информация явно выделена и описывается иными средствами, чем декларативная информация. Вместо логического вывода, характерного для логических моделей, в производственных моделях появляется вывод на знаниях. Вывод на такой базе знаний бывает *прямой* или *обратный*. В системе производствий с обратными выводами с помощью правил строится дерево «И/ИЛИ», связывающее в единое целое факты (посылки) и доказываемое (опровергаемое) утверждение. Оценка этого дерева на основании фактов, имеющихся в базе данных, и есть логический вывод. Оценка заключается в том, что необходимо найти ту посылку, наличие или отсутствие которой в наибольшей степени подтвердит или опровергнет рассматриваемое утверждение. В системе производствий с прямым выводом известна посылка, нужно получить результат.

Наиболее распространенными являются системы производствий с прямым выводом. Они состоят из базы правил, включающей набор производствий (правил вывода), базы данных, в которой содержится множество фактов, и интерпретатора для получения логического вывода. База данных и база правил составляют базу знаний, а интерпретатор соответствует механизму логического вывода.

Производственная модель чаще всего применяется в промышленных экспертных системах. Она привлекает разработчиков своей наглядностью, высокой модульностью, легкостью внесения дополнений и изменений и простотой механизма логического вывода.

Вместе с тем, продукционной модели присущ ряд недостатков. Основной причиной недостатков продукционных моделей является нехватка строгой теоретической основы. При задании модели предметной области в виде совокупности продукции нельзя быть уверенным в ее полноте и непротиворечивости. Так, при накоплении достаточно большого числа (порядка нескольких сотен) продукции они начинают вследствие необратимости дизъюнкций противоречить друг другу. В этом случае разработчики начинают усложнять систему, включая в неё модули нечёткого вывода или иные средства разрешения конфликтов, — правила по приоритету, правила по глубине, эвристические механизмы исключений, возврата и т. п. *Механизм исключений* означает, что вводятся специальные правила-исключения. Их отличает большая конкретность в сравнении с обобщёнными правилами. При наличии исключения основное правило не применяется. *Механизм возвратов* же означает, что логический вывод может продолжаться в том случае, если на каком-то этапе вывод привёл к противоречию. Просто необходимо отказаться от одного из принятых ранее утверждений и осуществить возврат к предыдущему состоянию.

Еще один недостаток продукционной модели — это несоответствие структуры знаний системы структуре знаний человека. В частности, структура базы знаний продукционной системы не позволяет описывать метазнания и свойственную человеческому мышлению нечеткую логику.

1.6. Фреймовые модели представления знаний

Фреймовая модель представления знаний основана на фреймовой теории, предложенной М.Минским в 1974 г., и представляющей собой систематизированную в виде единой теории психологическую модель памяти человека и его сознания. Важным моментом во фреймовой теории является понятие фрейма — однажды определенной структуры данных для представления некоторого концептуального объекта. Информация, относящаяся к конкретному фрейму, содержится в слоте. Все фреймы

объединяются в иерархическую структуру, интегрирующую в себе декларативные и процедурные знания. Данная структура отображает целостный образ знаний, которому свойственна иерархичность концептуального представления.

Основным отличием фреймовых моделей является жесткая фиксация структуры информационных единиц, которая называется протофреймом. В общем виде она выглядит следующим образом:

(Имя фрейма:

 Имя слота 1 (значение слота 1)

 Имя слота 2 (значение слота 2)

.....

 Имя слота K (значение слота K)).

Значением слота могут быть числа или математические соотношения, тексты на естественном языке или программы, правила вывода или ссылки на другие слоты данного фрейма или других фреймов. Таким образом фреймы объединяются в сеть. Свойства фреймов наследуются сверху вниз, то есть от вышестоящих к нижестоящим.

При конкретизации фрейма ему и слотам присваиваются конкретные имена и происходит заполнение слотов. Незаполненный фрейм называется *протофреймом*, а заполненный – *фреймом-экземпляром*. Переход от исходного протофрейма к фрейму-экземпляру может быть многошаговым за счет постепенного уточнения значений слотов. Связи между фреймами задаются значениями специального слота с именем «Связь». Таким образом, во фреймовых моделях в представлении знаний практически объединены все основные особенности моделей остальных типов.

Слот может содержать не только конкретное значение, но и имя процедуры, позволяющей вычислить его по заданному алгоритму, а также одну или несколько продукции (эвристик), с помощью которых это значение определяется. В слот может входить не одно, а несколько значений. Иногда

этот слот включает компонент, называемый *фасетом*, который задает диапазон или перечень его возможных значений. Фасет указывает также граничные значения заполнителя слота.

Помимо конкретного значения в слоте могут храниться процедуры и правила, которые вызываются при необходимости вычисления этого значения. Среди них выделяют *процедуры-демоны* и *процедуры-слуги*. Первые запускаются автоматически при выполнении некоторого условия, а вторые активизируются только по специальному запросу.

Совокупность фреймов, моделирующая какую-либо предметную область, представляет собой иерархическую структуру, в которую фреймы собираются с помощью родовидовых связей. На верхнем уровне иерархии находится фрейм, содержащий наиболее общую информацию, истинную для всех остальных фреймов. Фреймы обладают способностью наследовать значения характеристик своих родителей, находящихся на более высоком уровне иерархии. Эти значения могут передаваться по умолчанию фреймам, находящимся ниже них в иерархии, но если последние содержат собственные значения данных характеристик, то в качестве истинных принимаются именно они. Это обстоятельство позволяет без затруднений учитывать во фреймовых системах различного рода исключения.

Различают *статические* и *динамические* системы фреймов. В системах первого типа фреймы не могут быть изменены в процессе решения задачи, а в системах второго типа это допустимо.

Основным преимуществом фреймов как модели представления знаний является то, что она отражает концептуальную основу организации памяти человека, а также ее гибкость и наглядность. Однако разрозненные части информации, объединенные во фрейм, не могут быть выстроены в последовательность высказываний. Иначе говоря, языки описания знаний во фреймовой модели не являются языками, родственными естественным, а ближе к изобразительным средствам. Также недостатком фреймовой модели является отсутствие специального механизма управления выводом, в связи с

чем разработчики должны реализовать данный механизм с помощью присоединенных процедур.

ГЛАВА 2. МОДЕЛИ И МЕТОДЫ ОНТОЛОГИЧЕСКОГО ИНЖИНИРИНГА

2.1. Предпосылки появления онтологической модели представления знаний и перспективы использования

Основным результатом исследований в области искусственного интеллекта является не только разработка различного вида интеллектуальных систем, но и создание технологий, позволяющих быстро разрабатывать интеллектуальные системы, имеющие практическую ценность. Составляющими таких технологий являются

- формальная теория искусственного интеллекта;
- методы проектирования интеллектуальных систем;
- средства автоматизации проектирования;
- средства информационной поддержки разработчиков;
- средства компьютерной поддержки управления коллективной разработкой.

Анализ современных технологий искусственного интеллекта показывает, что наряду с впечатляющими достижениями они имеют целый ряд серьезных недостатков. К числу таких недостатков, в частности, относится отсутствие общего унифицированного решения проблемы семантической совместимости информационных систем, порожденное разнообразием применяемых разработчиками моделей, методов, средств представления знаний. Это обуславливает высокую трудоемкость создания комплексных интегрированных ИС. Наиболее остро проблема совместимости ИС проявляет себя при разработке web-ориентированных систем. В данной области широко известны работы Томаса Грубера, который

занимался исследованиями в части создания механизма взаимодействия баз знаний, обеспечивающего интерпретацию знаний, импортированных из разных источников, на уровне семантики. В рамках решения данной проблемы он предложил подход, согласно которому формальные модели баз знаний должны были максимально просты и легко понимаемы не только их интерпретаторами, но и всеми разработчиками. Доступность семантики для пользователей можно обеспечить за счет выбора соответствующего формата представления знаний. Однако для прикладных программ семантика должна быть представлена в формальной, машинно-обрабатываемой форме. Поэтому Грубер разделил описание знаний на две составляющие:

- онтологию, описывающую предметную область в виде иерархической структуры, содержащей описание всех концептов предметной области, их связей и правил, принятых в предметной области.
- каноническую форму, описывающую знания на языке логики предикатов.

Каждая интеллектуальная система, ядром которой является база знаний, может предоставлять несколько таких описаний, соответствующих различным областям хранящихся в ней декларативных знаний, и выступать, таким образом, как хранилище библиотеки онтологий. При этом библиотеке онтологий уже не обязательно быть интеллектуальной системой, достаточно просто предоставлять сервис по передаче онтологий по требованию.

Составление описания декларативного знания обычно требует большой работы и определенных навыков. Для обозначения этой работы, а также ее результата, Грубер ввел в обиход специальный термин «концептуализация». Описание он называл «спецификацией». Таким образом, онтология по Груберу определяется как *спецификация концептуализации*.

Введенное Грубером разделение спецификаций знаний на две составляющие (каноническую форму и онтологию) не очень удобно, т.к. приходится описывать одни и те же знания два раза. Современные языки

описания онтологий позволяют совместить эти формы спецификаций в единое целое. Сейчас под онтологией понимается любое описание декларативных знаний, сделанное на формальном языке и снабженное некоторой классификацией специфицируемых знаний, позволяющей человеку удобно воспринимать их. Любое такое описание должно включать в себя представление декларативных знаний в виде иерархии объектов (классов), только в этом случае это описание может считаться онтологией. Такой способ описания знаний объединяет в себе другие известные модели представления знаний. Таким образом, онтология – это иерархическая структура конечного множества понятий, описывающих заданную предметную область, такая, что:

- 2) структура представляет собой онтограф, вершинами которого являются понятия, а дугами – семантические отношения между ними;
- 3) понятия и отношения интерпретируются в соответствии с общезначимыми функциями интерпретации, взятыми из источников знаний заданной предметной области;
- 4) определение понятий и отношений выполняется аксиомами и ограничениями области действия;
- 5) формально онтограф описывается на одном из языков описания онтологий;
- 6) функции интерпретации и аксиомы описаны в некоторой подходящей формальной теории.

Перспективы использования онтологий при описании баз знаний можно подразделить на следующие категории:

- улучшение взаимодействия;
- унификация обмена данными;
- формализация процессов спецификации;
- повышение надежности и обеспечения многократности использования.

Улучшение взаимодействия связано с ожиданием уменьшения терминологической и концептуальной путаницы и неоднозначности понимания на основе осуществления следующих мероприятий:

- Создание в конкретной области человеческой деятельности (среде) унифицирующего нормативного ядра понятий, которое позволило бы достичь однозначного семантического толкования основных объектов и процессов этой области. Предполагается, что это нормативное ядро является семантической основой для порождения, переопределения и интерпретации новых понятий.
- Создание однозначно понимаемого множества отношений между понятиями нормативного ядра, допускающего исследование динамических и статических аспектов среды, влияния на нее различных факторов, вывод и планирование ситуаций.
- Обеспечение совместимости онтологий, разработанных различными коллективами путем широкого использования полисинонимии и ретранслируемости.
- Обеспечение возможности коллективной работы по согласованию и унификации онтологий и их нормативного ядра.

Унификацию обмена данными связывают с созданием интегрированных инструментальных программных средств, построенных на основе использования нормативного ядра и множества отношений. Эти средства должны обеспечивать возможность обмена данными для созданных или создаваемых на базе онтологий систем моделирования сред и выступают как средство межъязыкового общения, как своеобразный декларативный язык, на который переводятся другие языки и из которого перевод осуществляется в индивидуальный язык пользователя. Эти средства должны иметь в своем составе поддерживаемые, развивающиеся и доступные для использования извне библиотеки онтологий.

Формализацию процессов спецификации, повышения надежности и обеспечения многократности использования связывают с ролью онтологий,

которую они призваны играть для развития систем моделирования сред. Язык онтологии выступает в этом случае как средство спецификации таких систем и является декларативным. Роль онтологии зависит от степени выразительности, формализованности и других свойств декларативного языка онтологии. Повышение надежности систем моделирования связывают с возможностью и удобством полуформального и формального анализа декларативного описания на языке онтологии. При этом, говоря о формальном анализе, полагают, что описание допускает формальный вывод (доказательство) наличия тех или иных свойств среды. Обеспечение многократности использования предполагает наличие в онтологии метауровня, позволяющего настроить онтологию на конкретную задачу применения, определить степень ее пригодности для решения конкретной задачи и модифицировать или расширить ее, если это необходимо.

Наиболее характерной сферой применения онтологий в настоящее время является представление знаний в Интернете. Круг связанных с этим вопросов весьма широк и включает в себя мультиагентные системы, автоматическое извлечение знаний из текстов на естественном языке, поиск информации, интеллектуальное аннотирование, автоматическое составление авторефераторов и прочее.

2.2. Формальная модель онтологии

Онтология описывает знания о некоторой предметной области с помощью набора понятий (классов), связанных определенным набором отношений (свойств), а также набора функций, необходимых для ограничения интерпретации и использования понятий. Формальной моделью онтологии **O** является упорядоченная тройка вида:

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

где:

X – конечное непустое множество классов предметной области, которую представляет онтология **O**;

R – конечное множество отношений между классами;

F – конечное множество функций интерпретации (аксиоматизации), заданных на классах и/или отношениях онтологии **O**.

Классы онтологии представляют собой понятия предметной области. Формально класс интерпретируется как множество объектов предметной области – экземпляров классов. Например, для предметной области «Университет» такими классами являются «Факультет», «Группа», «Студент», «Дисциплина» и др.

Между классами могут быть установлены определенные связи, называемые свойствами. Формально свойства представляют собой бинарные отношения на классах. Базовыми типами свойств являются «класс-подкласс», «часть-целое», «экземпляр-класс», «причина-следствие», «похоже на» и т. п. Например, Студент_Группа, Группа_Факультет и т.д.

Роль функции интерпретации может играть словесное пояснение термина (аннотация), формула для вычисления значения термина, алгоритмическое описание, а также определение в виде логической формулы. Аксиомы используются для моделирования утверждений, которые всегда являются истинными. Аксиомы могут быть включены в онтологию для разных целей, например, для определения комплексных ограничений на аргументы отношений, для проверки корректности информации, описанной в онтологии, или для вывода новой информации.

Рассмотрим частные случаи онтологии.

Пусть $R = \emptyset$ и $F = \emptyset$. Тогда онтология **O** трансформируется в простой словарь, т.е. конечный список терминов:

$$O = V = \langle X, \emptyset, \emptyset \rangle.$$

Такая вырожденная онтология может быть полезна для спецификации, пополнения и поддержки словарей предметной области, но онтологии-словари имеют ограниченное использование, поскольку не вводят эксплицитно смысла терминов. Хотя в некоторых случаях, когда используемые термины принадлежат очень узкому (например, техническому)

словарю и их смыслы уже заранее хорошо согласованы в пределах определенного (например, научного) сообщества, такие онтологии применяются на практике. Известными примерами онтологии этого типа являются индексы машин поиска информации в сети Интернет.

Пусть $R = \{ \text{is_a} \}$ – отношение строгого порядка на множестве X , устанавливающее иерархию классов предметной области; $F = \emptyset$. Тогда онтология превращается в таксономию – иерархическую систему понятий

$$\mathbf{O} = \mathbf{T} = \langle X, \{\text{is_a}\}, \emptyset \rangle.$$

Отношение **is_a** имеет фиксированную заранее семантику и позволяет организовывать структуру понятий онтологии в виде дерева. Такой подход имеет свои преимущества и недостатки, но в общем случае является адекватным и удобным для представления иерархии понятий.

По своей функциональной полноте и степени формальности различают три вида онтологий: простая, полная (или строгая) и множество промежуточных или неполных онтологий.

Простая онтология – это такая онтология, в которой $R = \emptyset$ и $F = \emptyset$. Она служит (в основном) для однозначного восприятия научным сообществом понятий в соответствующей прикладной области.

Строгая или полная онтология ($R \neq \emptyset$ и $F \neq \emptyset$) – это такая онтология, в которой множества классов и отношений максимально полные, а к функциям интерпретации добавляются аксиомы, определения и ограничения. При этом описания всех компонент представлены на некотором формальном языке, доступном для их интерпретации компьютером. Схема формальной модели полной онтологии описывается четвёркой:

$$\mathbf{O} = \langle X, R, F, A(D, G) \rangle$$

где:

X – конечное непустое множество классов;

R – конечное множество отношений между классами;

F – конечное множество функций интерпретации, заданных на классах и/или отношениях;

A – конечное множество аксиом, которые используются для записи всегда истинных высказываний (определений и ограничений);

D – множество дополнительных определений понятий;

G – множество ограничений, определяющих область действия понятийных структур.

Полная онтология является формальным выражением концептуальных знаний о предметной области и представляет собой понимают базу знаний специального типа, которую можно разделять, отчуждать, и самостоятельно использовать в рамках рассматриваемой предметной области.

Множество промежуточных или неполных онтологий ($R = \emptyset, F \neq \emptyset$; $R \neq \emptyset, F = \emptyset$) возникает, когда для каждого концепта (или их большей части) добавлены аксиомы и определения. Одним из распространённых вариантов неполной онтологии является структура вида $R \neq \emptyset, F = \emptyset$, где множество F в явном виде отсутствует в предположении, что концепты $x \in X$ общеизвестны (определенены по умолчанию) либо (и) достаточно полно интерпретированы отношениями R .

2.3. Типы онтологий. Формальная модель онтологической системы

Выделение типов онтологий будет приведено на основе таких признаков, как универсальность и выразительность. Уровень универсальности определяет масштаб онтологии, а выразительность – детальность ее описания.

По уровню универсальности выделяют четыре типа онтологий:

- Онтологии верхнего уровня (метаонтологии);
- Онтологии, ориентированные на предметную область (предметные онтологии);
- Онтологии, ориентированные на конкретную задачу (онтологии задач);

- Прикладные онтологии.

Под *метаонтологией* (онтологией верхнего уровня) понимается самая общая онтология, которая является общей для всех областей знаний. Метаонтология описывает наиболее общие понятия и отношения, которые не зависят от предметных областей, например, «Объект», «Процесс», «Свойство», «Значение» и т.д. Считается, что такая онтология существует, и разные группы специалистов предъявляют свои варианты. Существует несколько крупных онтологий верхнего уровня: Cyc, DOLCE, SUMO. Метаонтологии связаны с мировоззрением, и, естественно, эта область близка к исследованиям в философии и лингвистике.

Предметная онтология описывает словарь, связанный с предметной областью за счет специализации терминов, введенных в метаонтологии. Предметная онтология содержит понятия, описывающие конкретную предметную область, отношения, семантически значимые для данной предметной области, и множество интерпретаций этих понятий и отношений. Предметная онтология обычно применяется для того, чтобы уточнить понятия, определённые в метаонтологии (если используется), и/или определить общую терминологическую базу предметной области. Предметные онтологии используются только внутри одной предметной области.

Онтология задач описывает концептуальную модель конкретной задачи. Такая онтология содержит в качестве понятий типы решаемых задач, а отношения специфицируют декомпозицию задач на подзадачи. Онтологии задач содержат наиболее специфичную информацию.

Прикладные онтологии описывают концепты, зависящие как от конкретной предметной области, так и от задач, которые в них решаются. Концепты в таких онтологиях часто соответствуют ролям, которые играют объекты в предметной области в процессе выполнения определенной деятельности.

Центральной идеей системно-онтологического подхода является разработка онтологических средств поддержки решения прикладных задач – полифункциональной онтологической системы. Под формальной моделью онтологической системы S_o понимают триплет вида:

$$S_o = \langle O^{meta}, \{O^{sub}, O^{task}\}, I \rangle,$$

где O^{meta} онтология верхнего уровня (метаонтология);

$\{O^{sub}, O^{task}\}$ – множество предметных онтологий O^{sub} и онтологий задач предметной области O^{task} ;

I – модель машины вывода, ассоциированной с онтологической системой S_o .

В модели S_o имеются три онтологические компоненты:

- метаонтология;
- предметная онтология;
- онтология задач.

Как указывалось выше, метаонтология оперирует общими концептами и отношениями, которые не зависят от конкретной предметной области. Тогда на уровне метаонтологии мы получаем интенсиональное описание свойств предметной онтологии и онтологии задач. Онтология метауровня является статической, что дает возможность обеспечить здесь эффективный вывод. Использование системы онтологии и специальной машины вывода позволяет решать в такой модели различные задачи. Расширяя систему моделей $\{O^{sub}, O^{task}\}$, можно учитывать предпочтения пользователя, а, изменяя модель машины вывода, вводить специализированные критерии релевантности получаемой в процессе поиска информации и формировать специальные репозитории накопленных данных, а также пополнять при необходимости используемые онтологии.

По степени выразительности выделяют следующие типы онтологий:

1. Контролируемый словарь – конечный список терминов, которые используются, чтобы пометить единицы информации так, чтобы они могли быть более легко восстановлены поиском. Контролируемые словари

уменьшают двусмысленность, врожденную от нормальных естественных языков, где тому же самому понятию можно дать различные имена и гарантировать последовательность.

2. Глоссарий, представляющий собой список терминов с их значениями. Значения описываются в виде комментариев на естественном языке. Это дает больше информации, поскольку люди могут прочесть такой комментарий и понять смысл термина. Интерпретации терминов могут быть многозначными.

3. Тезаурус – словарь терминов, в котором явно указаны отношения между терминами (род-вид, целое-часть, синонимические отношения и др.).

4. Таксономия – древовидная иерархия терминов, использующихся для классификации объектов. Эта разновидность онтологий включает точное определение отношения Подкласс_Класс (обозначаемого как *is_a*). В таких системах строго соблюдается транзитивность отношения *is_a*: если А является подклассом класса В, то каждый подкласс класса А также является подклассом класса В. Строгая иерархия классов необходима при использовании наследования для процедуры логического вывода.

5. Онтологии, в которых установлено отношение класс-экземпляр. Такие онтологии содержат на нижнем уровне конкретные объекты предметной области – экземпляры.

6. Онтологии, в которых установлены свойства типов данных – связи между характеристиками классов и значениями типов данных. Например, класс «Продукт» может иметь характеристику «цена» с типом данных «вещественное число». Свойства типов данных бывают особенно полезными, когда они определены на верхних уровнях иерархии и наследуются подклассами. Так, в потребительской иерархии класс «Продукт» может иметь свойство «цена», которое получат все его подклассы.

Большой выразительностью обладают онтологии, включающие ограничения на область значений свойств. Значения свойств берутся из некоторого предопределенного множества (целые числа, символьные

константы) или из подмножества концептов онтологии (множество экземпляров данного класса, множество классов). Можно ввести дополнительные ограничения на то, что может заполнять свойство. Например, для свойства «сделан из» класса «Предмет одежды» значения могут быть ограничены экземплярами класса «Материал».

7. Онтологии, содержащие описание классов с простыми логическими или математическими ограничениями на отношения.

В целом с необходимостью описывать более сложные факты выразительные средства онтологии (и ее структура) усложняются. Например, может потребоваться заполнить значение какого-либо свойства экземпляра, используя математическое выражение, основанное на значениях других свойств данного экземпляра или значениях свойств других экземпляров. Многие онтологии позволяют объявлять два и более класса дизъюнктивными (непересекающимися). Это означает, что у данных классов не существует общих экземпляров. Некоторые языки описания онтологий позволяют делать произвольные логические утверждения о концептах – аксиомы, а также фиксировать утверждения на языке логики предикатов первого порядка.

2.4. Характеристики качества онтологий интеллектуальных информационных систем

Цель создания онтологий – обеспечить поддержку деятельности по накоплению, разделению и повторному использованию знаний. Исходя из этой цели, существует ряд критериев качества, которым должна отвечать онтология:

1. Ясность – онтология должна ясно и однозначно передавать смысл введенных терминов. Определения должны быть объективными, хотя мотивация введения терминов может определяться ситуацией или требованиями вычислительной эффективности. Для объективизации определений должен использоваться четко

фиксированный формализм, при этом целесообразно задавать определения в виде логических аксиом.

2. Согласованность – все определения должны быть логически непротиворечивы, а все утверждения, выводимые в онтологии, не должны противоречить аксиомам.
3. Расширяемость – онтология должна быть спроектирована так, чтобы обеспечивать использование разделяемых словарей терминов, допускающих возможность монотонного расширения и/или специализации без необходимости ревизии уже существующих понятий.
4. Минимальное влияние кодирования. В онтологической системе должен быть реализован принцип совместного использования онтологий, который предполагает спецификацию онтологии на уровне представления, а не символьного кодирования. Запись такой спецификации на общепринятом и платформонезависимом языке описания онтологий можно передать для использования любому программному агенту.
5. Минимальные онтологические соглашения. Онтология должна содержать только наиболее существенные предположения о моделируемой предметной области, но их должно быть достаточно для описания знаний, которые предполагается совместно использовать.

Для обеспечения успеха при разработке, внедрении и использовании онтологии, ее оценка должна проводиться на протяжении всего жизненного цикла по заранее определенным критериям, зависящим от назначения онтологии и средств ее поддержки. Оценка онтологии включает в себя сбор информации об определенных её параметрах, проверку соответствия этой информации некоторым требованиям и оценку пригодности онтологии для конкретной задачи. Некоторые свойства онтологий не связаны с конкретной задачей, другие требуют оценки отношений между онтологией и её

предметной областью, окружением или особыми вариантами её использования и могут оцениваться исключительно в контексте сценария использования. Разнообразие потенциальных вариантов использования онтологий не позволяет создать универсальный набор критериев оценки. Таким образом, не существует одного метода оценки, применимого ко всем онтологиям. Несмотря на это, можно выделить некоторые методы оценки, необходимые для большинства онтологий. Оценка качества онтологии предполагает её оценку в качестве модели предметной области, пригодной для понимания человеком, модели для машинной обработки и онтологии, как части сложной программной системы. Главными характеристиками здесь являются следующие:

1. Разборчивость – оценивается пригодность онтологии к пониманию её человеком.
2. Точность – оценивается, насколько точно онтология представляет предметную область.
3. Мастерство – оценивается, насколько хорошо построена онтология, и в какой мере соблюдены оригинальные организационные решения.
4. Степень соответствия – оценивается, насколько подходит онтологическая модель под решаемую задачу.
5. Простота развертывания – оценивается, отвечает ли онтология требованиям системы, частью которой является.

Для обеспечения разборчивости недостаточно того, чтобы онтология была читаема специалистами-онтологами. Все целевые пользователи системы должны быть в состоянии интерпретировать её содержимое (экземпляры, классы, отношения и т.д.), важное для их деятельности. Разборчивость не предполагает прямого распознавания онтологии пользователями, однако документация на нее должна быть понятна всей целевой аудитории. Для этого может требоваться наличие множественных определений для одного понятия (например, на разных языках).

Разборчивость особенно важна для онтологий, используемых в качестве управляемого словаря, однако и для онтологий, применяющихся в качестве внутренней структуры программных систем, также желательно обеспечение разборчивости, так как поддержка онтологий осуществляется людьми, не всегда причастными к её созданию.

Точность отражает корректность описания предметной области как в аксиомах, так и документации к онтологии. Мастерство отвечает за аккуратность исполнения онтологии от наличия синтаксических ошибок до вопросов правильности реализации философского базиса онтологии.

Требования к соответствию задачам и развертываемости онтологии зависят от сценария её использования. Онтология является моделью конкретной предметной области, которая разрабатывается для конкретной цели. Следовательно, не существует правильной онтологии определенной предметной области. Онтология представляет собой абстракцию конкретной области и всегда имеет жизнеспособные альтернативы. Что именно будет включено в эту абстракцию, определяется в зависимости от задач приложения, для которого она разрабатывается, и от будущих потенциальных вариантов ее использования. Лучшее решение зависит от предполагаемого приложения и ожидаемых расширений. Поэтому оценка должна производиться с учетом требований к системе, частью которой является онтология.

2.5. Жизненный цикл создания онтологии

Жизненный цикл (ЖЦ) любой онтологии состоит из ряда процессов, в ходе которых онтология зарождается, специфицируется, адаптируется, развертывается, используется и поддерживается. Происходят эти процессы параллельно или последовательно, однажды за ЖЦ или повторяются несколько раз, частично зависит от того, как выполнялось создание онтологии. Поскольку онтологии создаются для разных задач, отдельные стадии жизненного цикла могут отсутствовать для одних онтологий и

присутствовать для других. Данный факт не позволяет создать единую общую модель ЖЦ онтологии с четко обозначенной последовательностью этапов ЖЦ. ЖЦ конкретных онтологий являются упрощенными частными случаями, зависящими от специфики создания конкретной онтологии. Оценка онтологий производится на всех стадиях ЖЦ, что позволяет установить, насколько онтология удовлетворяет требованиям последующей стадии. Рассмотрение упрощенного ЖЦ позволяет выделять стадии, общие для всех онтологий:

1. Стадия формирования требований.
2. Стадия онтологического анализа.
3. Стадия проектирования онтологии.
4. Стадия проектирования информационной системы, основанной на онтологии.
5. Стадия разработки онтологии.
6. Стадия разработки информационной системы, основанной на онтологии.
7. Стадия развертывания.
8. Стадия промышленной эксплуатации

1. *Стадия формирования требований.* На данной стадии рассматриваются и анализируются все возможные сценарии применения будущей онтологии, на основе анализа формулируются начальные требования. Как правило, сценарий использования понятен из задач онтологии. На начальном этапе требования могут быть представлены фрагментарно и касаться только отдельных задач. Одним из способов формулирования требований является использование вопросов проверки компетенции, т.е. вопросов, на которые должна отвечать онтология. Эти вопросы формулируются на естественном языке и соответствуют запросам, которые должна поддерживать онтология в выбранных сценариях использования. Результатом этапа формирования требований является документ, который должен отвечать на следующие вопросы:

1. Зачем нужна данная онтология? (причина появления, ожидаемая польза)
2. Каковы предполагаемые сценарии использования?
3. Какие группы пользователей должны быть в состоянии понимать определенные части онтологии?
4. Каков масштаб онтологии?
5. Есть ли существующие онтологии и стандарты, пригодные для использования?
6. Каковы вопросы компетенции?
7. Отражают ли вопросы компетенции все сценарии использования онтологии?
8. Каковы требования рабочей среды?
9. Какие ресурсы стоит принять во внимание при создании онтологии?
(Например, имеющиеся базы данных, модели данных, глоссарии, словари, схемы, таксономии, онтологии, стандарты, доступ к экспертам предметной области).

2. *Стадия онтологического анализа.* Задачей стадии онтологического анализа является выделение ключевых сущностей онтологии (экземпляров, классов и отношений между ними), а так же отождествление их с терминологией выбранной предметной области. Итоги этой работы, как правило, представляются в неформальном виде, пригодном для восприятия как для онтологов, так и для экспертов в предметной области (например, в виде таблиц и диаграмм). Результат онтологического анализ должен определять следующую информацию:

- Важные сущности в пределах предполагаемой предметной области.
- Важные характеристики сущностей, включая отношения между ними, неоднозначность описаний и свойства, важные для предметной области в рамках выбранного сценария использования.

- Терминологию, используемую для обозначения этих сущностей и предоставление достаточного количества контекстуальной информации для устранения неоднозначности многозначных терминов.

Результат анализа дает исходную информацию для проектирования и создания онтологии. Общая схема этапов онтологического анализа представлена на рис. 1

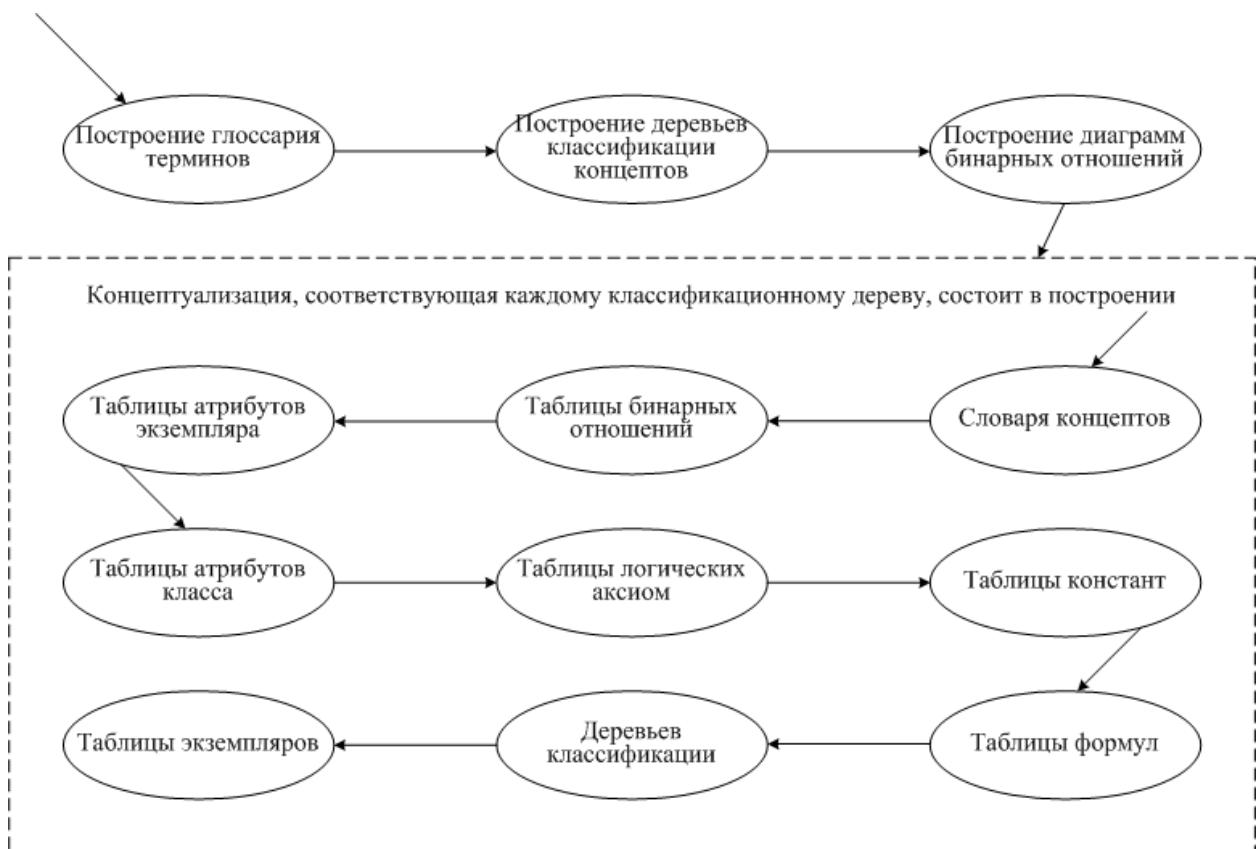


Рис. 1 – Стадия онтологического анализа

Следует заметить, что процесс построения онтологии здесь распадается на серию подпроцессов по созданию промежуточных представлений. При этом выполнение отдельных подпроцессов не последовательное, а определяется полнотой и точностью уже накопленных знаний.

Сначала строится глоссарий терминов, включающий все термины (концепты и их экземпляры, атрибуты, действия и т. п.), важные для предметной области, и их естественно-языковые описания. Понятия в онтологии должны быть близки к объектам (физическим или логическим) и

отношениям в интересующей предметной области. Наиболее часто это существительные (объекты) или глаголы (отношения) в предложениях, которые описывают предметную область. Когда глоссарий терминов достигает существенного объема, строятся деревья классификации концептов. Как правило, при этом используются отношения типа Класс_Подкласс и некоторые другие таксономические отношения. Таким образом, идентифицируются основные таксономии предметной области, а каждая таксономия дает в конечном счете онтологию.

Следующим шагом является построение диаграмм бинарных отношений, целью создания которых является фиксация отношений между концептами одной или разных онтологий. В дальнейшем эти диаграммы могут послужить исходным материалом для интеграции разных онтологий.

После построения представлений, фиксированных выше, для каждого дерева классификации концептов строятся:

1. Словарь концептов, содержащий все концепты предметной области, экземпляры таких концептов, атрибуты экземпляров концептов, отношения, источником которых является концепт, а также (опционально) синонимы и акронимы концепта.

2. Таблица бинарных отношений для каждого отношения, исходный концепт которого содержится в классификационном дереве. Для каждого отношения фиксируется его имя, имена концепта-источника и целевого концепта, инверсное отношение и т. п. характеристики.

3. Таблица атрибутов экземпляра для каждого экземпляра из словаря концептов. Основные характеристики здесь следующие: имя атрибута, тип значения, единица измерения, точность, диапазон изменения, значение «по умолчанию», атрибуты, которые могут быть выведены с использованием данного, формула или правило для вывода атрибута и др.

4. Таблица атрибутов класса для каждого класса из словаря концептов с аналогичными характеристиками.

5. Таблица логических аксиом, в которой даются определения концептов через всегда истинные логические выражения. Определение каждой аксиомы включает ее имя, естественно-языковое описание, концепт, к которому аксиома относится, атрибуты, используемые в аксиоме, логическое выражение, формально описывающее аксиому, и др.

6. Таблица констант, где для каждой константы указывается ее имя, естественно-языковое описание, тип значения, само значение, единица измерения, атрибуты, которые могут быть выведены с использованием данной константы, и т. п.

7. Таблица формул для каждой формулы, включенной в таблицу атрибутов экземпляра. Каждая таблица этого типа, помимо собственно формулы, должна специфицировать ее имя, атрибут, выводимый с помощью этой формулы, естественно-языковое описание, точность, ограничения, при которых возможно использовать формулу, и др.

8. Деревья классификации атрибутов, которые графически показывают соответствующие атрибуты и константы, используемые для вывода значения корневого атрибута и формулы, применяемые для этого. По сути дела, эти деревья используются для проверки того, что все атрибуты, представленные в формуле, имеют описания и ни один из атрибутов не пропущен.

9. Таблица экземпляров для каждого входа в словарь концептов. Здесь специфицируется имя экземпляра, его атрибуты и их значения.

Выходные параметры стадии онтологического анализа оцениваются в соответствии со следующими критериями:

1. Задокументированы ли все важные термины предметной области?
2. Выделены ли все сущности, важные в масштабе онтологии?
3. Согласны ли эксперты в предметной области с результатами онтологического анализа?
4. Является ли документация односмысленной в степени, достаточной для согласованного использования терминологии?

3. Стадия проектирования онтологии. На данной стадии разрабатывается проект онтологии – определяются принципы организации онтологии, проектируется структура онтологии, выбираются языки построения онтологии и язык запросов. Одним из способов организации онтологии является использование существующих онтологий, которые доступны в электронном виде и могут быть импортированы в используемую среду проектирования онтологии. Структура определяет разбиение онтологии на модули и то, как эти модули будут взаимодействовать между собой. Имеющиеся онтологии могут быть использованы в структуре в качестве модулей новой онтологии. Поведение модулей может быть описано по ответам на вопросы компетенции. Эти вопросы, специфичные для конкретных модулей, как правило, получают из вопросов для целой онтологии.

Следует отметить, что результаты проектировочных решений могут привести к конфликту требований выразительности и производительности онтологии. Данное противоречие можно разрешить путем создания отдельных справочных и операционных онтологий. Справочная онтология описывает предметную область во всей полноте, необходимой для решения задачи. Операционная онтология создается на её основе с возможным введением некоторых упрощений с целью увеличения производительности.

Оценка результатов фазы проектирования онтологий состоит в формировании документа, отвечающего на следующие вопросы:

1. Достаточны ли описательные возможности языка онтологии для удовлетворения требований к онтологии?
2. Достаточно ли выразителен язык запросов для формализации вопросов компетентности?
3. Поддерживает ли выбранный язык все необходимые возможности онтологии (например, если онтология оперирует вероятностями, язык должен описывать вероятностную информацию)?

4. Является ли каждый добавленный в онтологию класс или концепт подклассом или экземпляром класса верхнего уровня?
 5. Определены ли правила именования концептов и соблюдаются ли они?
 6. Требует ли проект создания нескольких отдельных онтологических модулей? Если да, то описывают ли модули в совокупности потребную предметную область.
 7. Описано ли в проекте, будут ли повторно использоваться созданные онтологии и как?
 8. Все ли модули онтологии имеют определенные (неформально) вопросы компетенции?
 9. Определено ли для каждого модуля, какие типы сущностей в нем представлены?
 10. Определено ли для каждого модуля, как он будет оцениваться и кто за это будет отвечать?
 11. Позволяет ли структура (конструкция) онтологии избегать добавления возможностей или содержимого, не относящегося к удовлетворению требований к онтологии?
4. *Стадия проектирования информационной системы, основанной на онтологии.* На данной стадии принимаются решения, влияющие на внедряемость онтологии в информационную систему. Оценка результатов стадии состоит в ответах на следующие вопросы:

1. Какие операции будут выполняться с использованием онтологий?
2. Какие компоненты будут выполнять эти операции?
3. Как бизнес-требования, разработанные на стадии определения требований, применимы к этим специфическим операциям и компонентам?
4. Будут ли и, если будут, то какие изменения и добавления в онтологии после развертывания системы?

5. Какие интерфейсы будут задействованы во внесении добавлений? Как будут тестироваться эти интерфейсы относительно измененной онтологии? Каким требованиям нужно будет отвечать?
6. Какие источники данных будут использоваться совместно с онтологией? Через какие интерфейсы будет происходить обмен информацией?
7. Как будет создаваться, оцениваться и поддерживаться онтологий? Какие для этого необходимы инструменты?
8. Если онтология будет иметь модульную структуру и/или создаваться распределенными разработчиками, как это будет поддерживаться?

5. *Стадия разработки онтологии.* Стадия включает четыре главных процесса:

- концептуальное моделирование,
- формализация вопросов компетенции,
- формальное моделирование,
- операционная адаптация.

Эти процессы обычно повторяются в цикле для отдельных модулей и для онтологии в целом. На практике они часто выполняются без четких границ между ними, тем не менее, важно понимать их концептуальные различия.

В процессе концептуального моделирования происходит доработка результатов онтологического анализа. Для каждого модуля происходит увязка терминологии с основными онтологическими концептами. Оценка результатов концептуального моделирования проводится на основе ответов на следующие вопросы:

1. Находятся ли в модели исключительно сущности выбранной предметной области?
2. Хорошо ли определены все концепты (например, не ссылаются сами на себя)?

3. Хорошо ли документирована интерпретация неопределенных экземпляров, классов и отношений?
4. Пригодна ли документация для понимания её экспертами предметной области?

Сценарии и вопросы компетенции формализуются на основе результатов концептуального моделирования. Вопросы, предназначенные для оценки результатов данного процесса, следующие:

1. Охватывают ли вопросы компетенции все сценарии использования?
2. Отражает ли формализованный вопрос цель вопроса изначального?

В процессе формального моделирования содержимое информационной модели записывается на каком-либо онтологическом языке, а затем конкретизируется аксиомами. Онтология, созданная или выбранная для повторного использования, оценивается по трем критериям:

- точность отображения предметной области;
- техническое совершенство (качество выполнения онтологии и то, насколько она отвечает требованиям, сформулированным в фазе онтологического проектирования);
- адекватность (насколько представление онтологии отвечает требованиям к её использованию).

В силу того, что оценка точности отображения предметной области зависит от понимания предметной области, оценка требует проверку содержания онтологии экспертами предметной области. Оценка технического совершенства происходит на основе оценки проектных и методологических решений стадии онтологического проектирования. В Поскольку проектирование онтологий – это молодая отрасль, существует слишком мало универсальных критериев оценки. Одним из способов проверки соответствия онтологии требованиям предметной области являются формализованные вопросы компетентности и сценарии. Соответствие онтологии требованиям предметной области также можно оценить путем проведения тестов.

В ходе операционной адаптации справочная онтология адаптируется к операционным задачам для получения операционной онтологии. Главным вопросом является, сможет ли новая онтология обеспечить требуемую производительность. Это может требовать упрощения онтологии или других оптимизационных процедур (например, реструктурирования). В некоторых случаях операционная онтология пишется на другом языке и с другой семантикой, нежели ссылочная.

6. Стадия разработки информационной системы, основанной на онтологии. На этом этапе происходит интеграция онтологии и других компонентов в подсистемы и систему в целом в соответствие с планом, разработанным на стадии проектирования. Стадия разработки и интеграции отнесена к ЖЦ онтологии по причине того, что, как правило, результат от использования онтологии получается только при её взаимодействии с другими компонентами информационной системы. Оценка результатов происходит, в частности, по ответам на следующие вопросы:

1. Успешно ли внедрена онтология?
2. Достигнут ли результат от внедрения онтологии, описанный в документации?

7. Стадия развертывания. На этом этапе онтология переходит от развития и интеграции к работе. Развертыванию обычно предшествует несколько циклов доработки, несмотря на это, она все равно может подвергаться дополнительным тестам перед внедрением. Оценка на этом этапе может проводиться с привлечением третьей стороны или включать в себя полную симуляцию работы системы. Целью таких испытаний является исключение негативного влияния внедрения онтологии на бизнес-процессы. Когда все испытания пройдены, онтология вводится в эксплуатацию и становится доступной для использования. Вопросы на этапе развертывания:

1. Отвечает ли онтология всем требованиям стадии разработки?
2. Существуют ли риски внедрения онтологии?

3. Использовались ли вопросы компетентности предыдущих этапов для создания регрессионных тестов?
4. Были ли проведены регрессионные тесты для оценки возможности снижения операционных показателей от внедрения системы? Если некоторое снижение прогнозируется, будет ли оно компенсировано положительным эффектом от внедрения онтологии?

8. *Стадия промышленной эксплуатации.* Этот этап фокусируется на поддержании имеющихся функций, а не на добавлении новых. Когда онтология (или ее версия) находится на стадии эксплуатации и технического обслуживания, происходит сбор информации о результатах оперативного использования онтологии. При выявлении проблем или фактов снижения операционных показателей могут проводиться микро-доработки для устранения возникших проблем. Одновременное выявление новых случаев использования, желаемых улучшений и новых требований, которое может произойти в течение того же периода использования, не следует рассматривать как часть технического обслуживания деятельности; скорее они являются предпосылками для разработки требований к будущей версии, расширению онтологии или созданию нового модуля. Вопросы к этапу промышленной эксплуатации:

1. Все ли регрессивные тесты пройдены успешно? Если нет, то какие меры принимаются?
2. Существуют ли проблемы функционирования системы? Если да, то вызваны ли они онтологией или проблемы в другом?
3. Если проблема в онтологии, может ли она быть решена без серьезного изменения онтологии?
4. Если проблема не может быть устранена без серьезной доработки онтологии, стоит ли продолжать её внедрение?

Как показывает анализ стадий ЖЦ онтологий, все они хорошо коррелируют с теми стадиями, которые выделены и используются при построении баз знаний. И это не случайное совпадение, а закономерность,

связанная с тем, что онтология — это, по существу, БЗ специального вида. Поэтому, как и в случае построения баз знаний, здесь используется концепция быстрого прототипирования, а специфика проявляется в тех конкретных процессах, которые реализуют рассмотренные выше процедуры.

При этом:

- планирование выполняется до начала собственно разработки;
- контроль и гарантии качества осуществляются в процессе разработки;
- большая часть операций по накоплению знаний и их оценке выполняется на стадии концептуализации для того, чтобы предотвратить распространение ошибок на фазу реализации;
- интеграция не должна рассматриваться как интеграция на стадии реализации. Напротив, она выполняется в процессе разработки.

Как правило, онтология в процессе своего ЖЦ проходит через выделенные стадии более одного раза. Идентификация стадий ЖЦ онтологии позволяет кластеризовать действия вокруг целей, входов и выходов узнаваемого типа. Более того, модель ЖЦ наглядно демонстрирует зависимость одних стадий ЖЦ от других, например, качество онтологии напрямую зависит от того, насколько грамотно были сформулированы требования к ней. Зависимости между стадиями инвариантны для всех онтологий, несмотря на их различия между собой.

2.6. Особенности проектирования предметных онтологий

Предметная онтология является формальным выражением концептуальных знаний о предметной области и по своей значимости сопоставима с базой знаний интеллектуальной информационной системы. Онтология определяет общеупотребительные, семантически значимые «понятийные единицы знаний», которыми оперируют исследователи и разработчики знаниеориентированных информационных систем. Предметная

онтология – это концептуальная модель реального мира и её компоненты должны отражать эту реальность.

Пусть $\mathbf{O} = \langle X, R, F \rangle$ – предметная онтология, где:

X – множество классов предметной области, которую представляет онтология \mathbf{O} ;

R – множество свойств онтологии;

F – множество функций интерпретации (аксиоматизации), заданных на классах и/или свойствах онтологии \mathbf{O} .

Построение множества X считается наиболее важным моментом при разработке онтологии предметной области. Оно должно быть обязательно не пустым. Для хорошо проработанных предметных областей за основу элементов множества X может быть взято содержимое подходящих словарей. В противном случае следует составить полный список терминов, в котором указать

- чем является каждый термин – понятием-классом предметов или конкретным понятием;
- указать для каждого термина возможные существенные отношения с другими терминами из списка;
- описать возможные существенные свойства понятий.

Известно, что в любой предметной области существуют термины-синонимы. Для них в онтологии отводится только одно понятие, в аксиомах которого может быть указан синонимический ряд терминов. Другими словами, синонимы одного и того же понятия не представляют различные классы. Далее уточняется и определяется окончательный список классов-понятий, имена которых будут входить в разрабатываемую онтологию и являться вершинами онтографа. При этом следует придерживаться единых правил присваивания имён понятиям и их свойствам. В результате должен быть получен полный список существенных для заданной предметной области (и предполагаемых приложений) понятий и их машинно-интерпретируемых формулировок.

Следующим шагом является определение свойств онтологии и построение онтологического графа. Построение онтографа является специальным видом классификации понятий предметной области – онтологической классификацией. В процессе соотнесения классов и построения иерархии следует учитывать, что:

- прямые подклассы в иерархии должны располагаться на одном уровне обобщения;
- класс может быть подклассом нескольких классов, и тогда он может наследовать свойства от всех этих классов;
- если класс имеет только один прямой подкласс, то, возможно, при моделировании допущена ошибка или онтология неполная;
- если у данного класса есть более 7-10 подклассов, то, возможно, необходимы дополнительные промежуточные классы.

В заключение данного подэтапа следует соотнести разработанные классы и их иерархии с результатами анализа предметной области. В частности, уточнить зависимости для конкретных пар экземпляров.

Множество аксиом онтологии состоит из множества определений и множества ограничений на понятия. Определения записываются в виде тождественно истинных высказываний, которые могут быть взяты, в частности, из словарей предметной области. В них могут быть указаны дополнительные взаимосвязи между классами. Ограничения – это аксиомы, которые ограничивают множество экземпляров, которые могут являться членами класса.

Следует учесть, что из полного списка отобранных в онтологию терминов не все представляют понятия. Существуют термины, которые соответствуют свойствам определённых классов-понятий. Такие свойства следует привязать к описанию самого общего класса, обладающего ими. А подклассы этого класса будут наследовать указанное свойство. Свойства понятий имеют определённые значения, такие как тип значений, мощность значений, разрешённые значения (для данного класса) и другие. Например,

мощность значений можно описать: с единичной мощностью, мощностью без ограничений и мощностью с некоторым допустимым интервалом.

На основе построенных множеств можно синтезировать концептуальную модель предметной области, получить формальное описание разработанной онтологии на одном из языков описания онтологий, а также графическое представление онтографа. При этом следует помнить, что в настоящее время не существует единственного правильного способа или методологии разработки онтологий. Онтология всегда отражает взгляд аналитика, т.е. всегда субъективна.

2.7. Отображение онтологий

Повторное использование существующих онтологий на сегодняшний день стало стандартным этапом в процессе разработки новой онтологии. Практически никто не ведет разработку онтологий «с нуля». Причем при разработке конкретного приложения чаще всего используется не одна существующая онтология, а их комбинация.

Интеграция онтологий представляет собой деятельность по созданию новой онтологии или фрагмента онтологии из двух и более исходных онтологий. Задача отображения онтологий возникла из необходимости интеграции онтологий, разработанных независимо друг от друга и имеющих, таким образом, свой собственный словарь. Также отображение онтологий является неотъемлемой частью большинства задач согласования онтологий, таких как выравнивание онтологий, модификация одной онтологии для достижения однородности с другой и так далее.

Отображение онтологий – это установление соответствия между концептами нескольких онтологий, или, другими словами, нахождение семантических связей подобных элементов из разных онтологий. Две (или более) онтологии могут по-разному описывать одну и ту же предметную область или близкие предметные области с точки зрения разных сообществ. Онтология задаёт подразумеваемую семантику для понятий предметной

области и определят онтологический контекст, в котором работает сообщество.

С наиболее общей точки зрения важность задачи отображения онтологий обусловлена тем фактом, что установление факта подобия сущностей в разных онтологиях означает извлечение из этих онтологий дополнительных знаний. Разные контексты использования онтологий, созданных разными сообществами, отражаются на особенностях подходов к спецификации понятий. В результате, семантика понятий в контекстах, описанных разными онтологиями, может быть сходной при различных подходах к описанию их структуры: составу, ограничениям и степени детализации. Проблема отображения онтологий заключается в том, что, во-первых, сущности, имеющие одинаковые имена, могут иметь разный смысл, а во-вторых, сущности, имеющие одинаковый смысл, могут иметь разные имена.

Отображение онтологий разделяется на две подзадачи:

1. Локальное отображение сущностей, подразумевающее независимое установление соответствий между двумя сущностями рассматриваемых онтологий.

2. Глобальное отображение сущностей, под которым подразумевается пересмотр локальных отображений с учетом отображений всех остальных элементов.

Современные методы установления отображения онтологий носят междисциплинарный характер. Выделяют лингвистические, статистические, структурные и логические методы. Для обеспечения максимальной точности отображения сущностей используют все четыре метода.

Лингвистические методы определяют сходство между сущностями исходной и целевой онтологии на основе сравнения их имен путем оценки количества совпадающих символов, нахождения общих частей слов и др., или на основе анализа синонимичных терминов. Для выявления

синонимичных терминов могут использоваться существующие словари общей и профессиональной лексики, тезаурусы.

Структурные методы включают в себя:

1. Анализ внутренней структуры исходной и целевой онтологии. Сущностей с похожими областями определения и областями значений может быть достаточно много, поэтому данные методы используются только для формирования кластеров сходных понятий и требуют сочетания с другими методами.

2. Анализ внешней структуры, включающий анализ сходства по иерархическим связям и анализ сходства по перекрестным связям.

Рассмотрим анализ сходства по иерархическим связям. Оценка схожести двух сущностей двух онтологий может быть основана на позициях данных сущностей в иерархии классов. Если две сущности двух онтологий схожи, то их «соседи» также как-то схожи. Такое утверждение может использоваться по-разному и порождает ряд возможных критериев (признаков) для сходства двух сущностей:

- их прямые супер-сущности (или все супер-сущности) уже являются схожими;
- их сущности-братья (или все их сущности-братья) уже являются схожими;
- их прямые сущности-потомки (или все их сущности-потомки) уже являются схожими;
- все их сущности-листья (сущности, не имеющие потомков, находящиеся в дереве, корнем которой является рассматриваемая сущность) уже являются схожими;
- все (или большинство) сущности на пути от корня к рассматриваемой сущности уже являются схожими.

Определение сходства между сущностями может быть основано также на анализе перекрестных связей сущностей. Если класс A_1 связан с классом B_1 свойством типа R_1 в одной онтологии, а класс A_2 связан с B_2 свойством

типа R_2 в другой онтологии, и если известно, что B_1 и B_2 – схожи, R_1 и R_2 – схожи, можно предположить схожесть A_1 и A_2 . Подобным образом можно говорить и сходстве типов свойств R_1 и R_2 , если известно, что A_1 и A_2 – схожи, B_1 и B_2 – схожи.

Для оценки экстенсионального соответствия классов используются существующие экземпляры классов. Для установки соответствия между сущностями используются диагностические правила, устанавливающие эквивалентность классов и отношение включения подкласса в класс. Анализ экстенсионала позволяет также идентифицировать классы-роли, когда возникает два разных класса для описания одного экстенсионала.

Логический анализ основан на выявлении родовых классов сопоставляемых классов и анализе наложенных на них ограничений. Например, в одной онтологии может существовать класс «Микро-компания», который является видовым классом для класса «Компания» с наложенным ограничением на «Число сотрудников» <5 . В другой онтологии может существовать класс «Малое предприятие», который является видовым классом для класса «Фирма» с наложенным ограничением на «Число работников» <10 . При анализе соответствия между классами «Микро-компания» и «Малое предприятие» выявляются родовые классы «Компания» и «Фирма». При наличие информации о соответствие данных классов производится сравнение ограничений наложенных на данные родовые классы. Для этого сравниваются свойства классов «Число сотрудников» и «Число работников», если данные свойства схожи, то проводится сравнение наложенных ограничений <5 (сотрудников) и <10 (работников). В результате делается заключение, что «Микро-фирма» \subset «Малое предприятие». Ограничением данного метода является потребность в «якорях» – сущностях, которые либо заведомо эквивалентны в двух сопоставляемых онтологиях, либо являются разделяемыми сущностями в некоторой сторонней онтологии. После получения локальных соответствий между сущностями определяется глобальное соответствие между сущностями.

Практические рекомендации по расстановке приоритетов между результатами различных способов локального анализа:

- При наличие баз знаний, включающих в себя экземпляры отображаемых онтологий, приоритетное значение имеют результаты экстенсионального анализа.
- При наличие «якорей» в отображаемых онтологиях приоритетное значение имеют результаты логического анализа.

Однако следует помнить, что результаты любого анализа следует согласовывать с результатами, полученными с использованием других видов анализа. Особенно важно такое согласование при установке соответствия между классами ролями, исполнители которых (экстенсионал) могут выполнять одновременно несколько ролей.

2.8. Онтологии как основа семантического веба

Количество информации, которую создает мировое сообщество, растет с каждым годом. Открытость информационного поля теоретически обеспечивает свободный и быстрый доступ к данным. Однако у такой всеобщей доступности есть и обратная сторона – чтобы получить информацию, ее нужно сначала найти.

Получение информации об интересующем объекте в подавляющем большинстве случаев сводится к использованию интернет ресурсов поисковых систем (ПС). Поиск информации поисковым роботом представляет собой процесс выявления в индексированном множестве ПС релевантных документов, т.е. таких, которые удовлетворяют заранее определенному запросу. Основная задача состоит в том, чтобы на конкретный запрос пользователя ПС провела обработку информации с последующим ранжированием найденных веб-ресурсов по их релевантности. Пользователь не может описать системе признаки искомого объекта, поскольку принцип поиска ПС базируется на тексте и ключевых словах. Фактически пользователю сложно найти данные, о которых он еще не знает,

а для их получения необходимо ввести в строку запроса информацию, содержащуюся в ответе. Приходится различать формальную релевантность и содержательную релевантность, причем, если формальная релевантность, повторяющая в листе выдачи ПС форму запроса, но не передающая изначально заданной содержательной сути сегодня достижима, то реализация содержательного соответствия документа смыслу запроса является в большинстве случаев нерешенной. Таким образом, основной проблемой нахождения смыслового соответствия документа пользовательскому запросу является разработка и реализация подходов, основанных на семантическом поиске.

Семантический поиск является одним из методов информационного поиска и представляет собой процесс поиска документов по их смысловому содержанию. Основой семантического поиска служат заранее установленные отношения между символами и объектами, которые они обозначают. Можно выделить два основных вида семантического поиска. Полнотекстовый поиск – поиск по всему содержимому документа с использованием предварительно построенных индексов. Поиск по метаданным – это поиск по неким атрибутам документа, которые описывают определенные объекты, поддерживаемые системой. Например, автор, адрес, название организации и т. д. Именно использование метаданных сегодня широко применяется в относительно новой концепции развития интернет под названием Семантический веб.

Концепцию Семантического веба (Semantic Web) выдвинул Тим Бернерс-Ли, один из основоположников World-Wide Web и председатель WWW-консорциума (W3C) на международной конференции XML-2000, прошедшей в 2000 году в Вашингтоне. Основная идея этого проекта заключается в организации такого представления данных в сети, чтобы допускалась не только их визуализация, но и их эффективная автоматическая обработка программами разных производителей. Путем таких радикальных преобразований концепции уже традиционного Веб-пространства

предполагается превращение его в систему семантического уровня. По замыслу создателей семантический веб должен обеспечить «понимание» информации компьютерами, выделение ими наиболее подходящих по тем или иным критериям данных, и уже после этого – предоставление информации пользователям. При автоматической обработке информации в рамках семантического веба взаимодействующие друг с другом сервисы на основе анализа смысловых связей между объектами и понятиями, хранящимися в сети Интернет, должны отбирать лишь ту информацию, которая будет реально полезна пользователям.

Таким образом, семантический веб – это расширение существующей Веб-сети, в котором информации приписывается точное, формально определенное значение, что дает возможность компьютерам «понимать» хранящуюся в Сети информацию и обрабатывать ее на семантическом уровне. Семантический веб формируется на базе Веб-сети путём стандартизации представления информации в виде, пригодном для машинной обработки.

Основной акцент концепции семантического веба делается на работе с метаданными, однозначно характеризующими свойства и содержание веб-ресурсов, вместо используемого в настоящее время текстового анализа документов. Эта концепция была принята и продвигается Консорциумом W3C. Для ее внедрения предполагается создание сети документов, содержащих метаданные о веб-ресурсах. Тогда как сами ресурсы предназначены для восприятия человеком, метаданные используются поисковыми роботами (агентами) для проведения однозначных логических заключений о свойствах этих ресурсов.

Семантический веб в математической форме представляет собой разновидность графа, где роль вершин выполняют понятия базы знаний, а направленные дуги задают отношения между ними. Таким образом, строится семантическая сеть, которая отражает семантику предметной области в виде понятий и отношений. Идея состоит в том, чтобы глобальной семантической

сетью было подмножество систем, которые замкнуты на специфичных путях достижения достаточного удобства для агентов.

Семантический веб основан на следующих принципах проектирования:

- 1) обеспечение доступности в вебе структурированных и полуструктурированных данных, представленных в стандартных форматах;
- 2) обеспечение доступности в вебе не только наборов данных, но и отдельных элементов данных и отношений между ними;
- 3) описание предполагаемой семантики таких данных с помощью формализма, обеспечивающего возможность машинной обработки этой семантики.

Данные принципы проектирования реализованы с помощью следующих семантических технологий:

1. В качестве модели данных для описания объектов и отношений между ними используются размеченные графы. Объекты представляются как вершины графа, а отношения между ними – как дуги. В качестве формализма для представления таких графов используется фреймворк описания ресурсов RDF (Resource Description Framework).
2. Для идентификации отдельных элементов данных и отношений между ними, которые включаются в наборы данных, используются веб-идентификаторы URI (Uniform Resource Identifier). Использование веб-идентификаторов отражено в стандарте языка RDF. В последних версиях семантического веба для идентификации ресурсов используется IRI (Internationalized Resource Identifier) – международный идентификатор ресурса, представляет собой Unicode-строку и соответствует синтаксису, определенному в стандарте URI.
3. В качестве модели данных, позволяющей формально представить предполагаемую семантику данных, используются онтологии.

Данная модель представляется с помощью таких формализмов, как язык RDF Shema и язык веб-онтологий OWL (Web Ontology Language), в которых URI (IRI) используются для идентификации терминов и их свойств.

Существование стандартов для описания данных (RDF) и их атрибутов (RDF Shema) позволяет создавать инструменты обработки информации из многочисленных источников. То, насколько глубоко различные приложения могут обмениваться данными и использовать их, принято называть синтаксическим взаимодействием сетей. Чем более стандартизованными и распространенными являются эти инструменты работы с данными, тем выше степень синтаксического взаимодействия сетей.

Синтаксическое взаимодействие сетей требует определенного преобразования между терминами, для чего, в свою очередь, необходим контентный анализ. Два поисковых агента могут использовать различные идентификаторы для обозначения одного и того же понятия и им необходимо объяснить, что два конкретных термина используются ими для обозначения одного и того же. Такой контентный анализ требует формальных и подробных спецификаций моделей доменов, которые определяют используемые термины и их связи. Подобные формальные модели доменов принято называть онтологиями. Они определяют модели данных в терминах классов, подклассов и свойств. Проще говоря, онтология – это документ, формально задающий отношения между терминами.

Наиболее типичными видами онтологий в вебе являются таксономия и набор правил вывода. Таксономия определяет классы объектов и отношения между ними. Например, понятие адрес может быть определено как разновидность понятия местонахождение, а код города можно задавать применительно лишь к местонахождениям и так далее. Большое количество отношений между инди видами можно задать путем приписывания классам определенных свойств и позволяя подклассам наследовать эти свойства. Правила вывода, задаваемые в онтологиях, дают еще больше возможностей.

В рамках онтологии можно записать такое правило: «Если объект А соответствует некоторому объекту В, а в объекте С фигурирует объект А, то этому объекту С тоже соответствует объект В». Поисковый робот не «понимает» в полном смысле этого слова ничего из всей этой информации, но теперь он уже может манипулировать терминами гораздо более эффективно с тем, чтобы стать полезным и осмысленным для пользователя.

Онтологический язык Web (Web Ontology Language), рекомендуемый W3C, добавляет больше словарных возможностей для описания свойств и классов, чем RDF или схема RDF. В частности, он позволяет описывать связи между классами, мощность множества, равенство, более богатую типологию свойств и их характеристики.

Рабочей группой W3C по доступу к данным разработан язык запросов SPARQL. SPARQL имеет SQL-подобный синтаксис, определяет запросы в терминах шаблонов графа, которые сравниваются с направленным графиком, представляющим данные RDF. SPARQL предоставляет возможности для запроса необходимых и необязательных шаблонов, а также для их объединения и разделения. Результат сравнения также может быть использован для конструирования нового графа RDF с использованием отдельного шаблона. Используя такие точки доступа SPARQL, агенты могут запрашивать удаленные RDF данные и формировать новые RDF графы без какой-либо локальной обработки.

Одним из первых серьезных и популярных проектов, основанным на принципах семантической паутины, стал проект «Дублинское ядро», реализуемый инициативной организацией Dublin Core Metadata Initiative (DCMI). Это открытый проект, цель которого разработать стандарты метаданных в формате RDF, независящие от платформ и подходящие для широкого спектра задач.

В то время как совокупность ресурсов и их метаданных можно считать статической частью семантической паутины, ее динамическую часть представляют так называемые семантические веб-сервисы – законченные

элементы программной логики с однозначно описанной семантикой, доступные через интернет и пригодные для поиска, композиции и выполнения. Консорциум W3C предполагает использование для описания веб-сервисов тех же языков разметки, что и для статической части семантической паутины, а также онтологии OWL-S, описывающей базовую терминологию предметной области. Онтология OWL-S состоит из четырех онтологий – онтологии сервиса, онтологии модели сервиса, онтологии процесса и онтологии базы.

Потенциальная выгода от использования семантических веб-сервисов заключается в возможности автоматического поиска программными агентами подходящих сервисов для решения поставленных задач. Тем не менее, сложность этой задачи в ее общей формулировке пока позволяет добиваться некоторых положительных результатов только в узкоспециализированных отраслях, явным образом выигрывающих от внедрения сервисо-ориентированной архитектуры (SOA), например, в интеграции корпоративных приложений.

Несмотря на очевидную актуальность Semantic Web существуют сложности ее практической реализуемости. Во-первых, необходимость описания метаданных приводит к дублированию информации. Правда, этот недостаток семантической паутины был главным толчком к созданию микроформатов, с помощью которых можно семантически размечать сведения о разнообразных сущностях непосредственно в коде HTML или XHTML. Во-вторых, в семантической паутине для получения ответа на некоторые вопросы совсем не обязательно переходить по ссылке на сайт. Доля поискового трафика на сайты может значительно снизиться, т.к. поисковые системы будут сами отбирать и предоставлять нужную пользователю информацию. Сответственно отпадает необходимость посещать сайт, на котором публикованы материалы и реклама, а значит, коммерческая выгода от привлечения пользователей на сайт уменьшается в разы. Здесь же можно сказать о том, что сохранение анонимности или же

авторских прав на текстовую информацию становится весьма проблематичным. Полнотекстовый семантический поиск является альтернативным подходом к концепции семантической паутины. В настоящее время разрабатываются алгоритмы, которые самостоятельно анализируют содержание интернета и переводят его из текстового представления в объектное.

ГЛАВА 3. ДЕСКРИПТИВНЫЕ ЛОГИКИ КАК ФОРМАЛЬНЫЕ МОДЕЛИ ОНТОЛОГИЙ

3.1. Базовые формализмы дескрипционных логиках

Построение, использование и эволюция онтологий в значительной степени зависит от их семантических свойств и инструментов логического вывода. Важное значение имеют при этом формальные модели описания онтологий и языки представления онтологии, которые бы поддерживали эту модель. Дескрипционные логики (ДЛ) являются семейством языков представления знаний, которые могут быть использованы для записи знаний предметной области формальным способом. ДЛ описывают знания прикладной проблемной области, вначале вводя подходящие понятия (его терминологию), а затем используя эти понятия для точного описания свойств объектов и экземпляров. Ключевой особенностью ДЛ, отличающей их от предшественников, таких, как семантические сети и фреймы, является то, что это логики, т.е. формальные языки с хорошо определенной семантикой. Другой их отличительной особенностью является наличие логического вывода, который позволяет выявить неявно представленные знания из знаний, которые явно содержатся в базе знаний. Разрешимость и сложность задач вывода зависит от выразительной силы конкретной ДЛ. С одной стороны, очень экспрессивные ДЛ сталкиваются с задачами вывода высокой сложности, которые могут быть даже неразрешимыми. С другой стороны, очень слабые ДЛ (с эффективными процедурами вывода) могут не обладать достаточной выразительностью для представления важных понятий конкретной прикладной задачи. Определение баланса между выразительностью ДЛ и сложностью их задач вывода является одной из наиболее важных проблем в исследованиях 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). Концепты используются для описания классов некоторых объектов, например, «Человек», «Женщина». Роль используются для описания двуместных отношений между объектами, например, на множестве людей имеется двуместное отношение «являться родителем».

Концепты и роли ДЛ представляют собой инструмент для записи знаний об описываемой предметной области. Эти знания подразделяются на общие знания о понятиях и их взаимосвязях (так называемые *интенсиональные знания*) и знания об индивидуальных объектах, их свойствах и связях с другими объектами (так называемые *экстенсиональные знания*). Первые более стабильны и постоянны, тогда как вторые более подвержены модификациям. В соответствии с этим делением знания, записываемые с помощью ДЛ, подразделяются на:

- набор терминологических аксиом или ТВоХ. ТВоХ описывает словарь интересующей предметной области, содержащий концепты и роли;
- набор утверждений об индивидах или АВоХ. АВоХ содержит утверждения о конкретных представителях (экземплярах) в терминах словаря предметной области.

Вместе они составляют так называемую базу знаний $B = \text{ТВоХ} \cup \text{АВоХ}$. Язык для построения описаний является характерной особенностью каждой системы ДЛ, и различные системы различаются их языком описания. Дескриптивный язык имеет теоретико-модельную семантику. Поэтому утверждения в ТВоХ и в АВоХ могут отождествляться с формулами в логиках первого порядка или, в некоторых случаях, незначительными их расширениями. Системы ДЛ позволяют не только описать терминологию и утверждения, но также предоставляют возможности по выполнению логического вывода с их помощью.

3.2. Синтаксис и семантика 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.

Для того чтобы задать какую-либо ДЛ, необходимо задать ее синтаксис и семантику. Синтаксис описывает, какие выражения (концепты, роли, аксиомы и т.п.) считаются правильно построенным в данной логике. Семантика указывает, как интерпретировать эти выражения, т.е. придает им формальный смысл.

Пусть $AC = \{A_1, \dots, A_m\}$ и $AR = \{R_1, \dots, R_n\}$ – конечные непустые множества атомарных концептов и атомарных ролей. Множество концептов логики ALC задается следующим индуктивным определением.

- выражения \top (Thing, истина) и \perp (NoThing, ложь) – концепты;
- всякий атомарный концепт A является концептом;
- если C – концепт, то выражение $\neg C$ является концептом и называется *дополнением* концепта C ;
- если C и D – концепты, выражения $C \sqcap D$ и $C \sqcup D$ являются концептами и называются *пересечением* и *объединением*;
- если C – концепт, а R – атомарная роль, то выражения $\exists R.C$ и $\forall R.C$ – концепты;
- никакие другие выражения не являются концептами.

В дальнейшем для формулировки синтаксиса будем использовать более краткую запись. Так, синтаксис для концептов логики ALC в этой записи выглядит следующим образом.

$$\top | \perp | A | \neg C | C \sqcap D | C \sqcup D | \exists R.C | \forall R.C,$$

где A – атомарный концепт, R – атомарная роль, C, D – произвольные концепты.

Семантика логики ALC задается с помощью понятия интерпретации. Интерпретация есть пара $I = (\Delta, \cdot^I)$, состоящая из непустого множества Δ ,

называемого областью данной интерпретации (доменом), и интерпретирующей функции \cdot^I , которая сопоставляет:

- каждому атомарному концепту $A \in AC$ – произвольное подмножество $A^I \subseteq \Delta$;
- каждой атомарной роли $R \in AR$ – произвольное подмножество $R^I \subseteq \Delta \times \Delta$.

Если пара экземпляров принадлежит интерпретации некоторой роли R , т.е. $(e, d) \in R^I$, то говорят, что экземпляр d является R -последователем экземпляра e .

Интерпретирующая функция распространяется на множество всех концептов логики ALC индукцией по построению концепта:

- \top интерпретируется как весь домен: $\top^I = \Delta$;
- \perp интерпретируется как пустое множество: $\perp^I = \emptyset$;
- дополнение концепта интерпретируется как дополнение множества: $(\neg C)^I = \Delta \setminus C^I$;
- 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 $\exists R.C$ интерпретируется как множество тех экземпляров, у которых имеется R -последователь, принадлежащий интерпретации концепта C :

$$(\exists R.C)^I = \{e \in \Delta \mid \text{существует } d \in \Delta \text{ такой, что}$$

$$(e, d) \in R^I \text{ и } d \in C^I\};$$

- выражение $\forall R.C$ интерпретируется как множество тех экземпляров, у которых все R -последователи принадлежат интерпретации концепта C :

$$(\forall R.C)^I = \{e \in \Delta \mid \text{для всех } d \in \Delta \text{ таких, что } (e,d) \in R^I, \text{ выполнено } d \in C^I\}$$

Пусть домен Δ – множество людей. Атомарным концептам Женщина, Мужчина, Родитель сопоставим соответственно множество женщин, множество мужчин и множество родителей. Атомарную роль иметь_ребенка интерпретируем двуместным отношением, связывающим всякого родителя с его ребенком. Другими словами, пара (e,d) принадлежит отношению иметь_ребенка¹, если d является ребенком e . Тогда можно дать интерпретацию следующим концептам:

- Женщина \sqcup Мужчина – множество всех людей;
- Женщина \sqcap Родитель – множество матерей;
- \exists иметь_ребенка.Т – множество людей, кто имеет хотя бы одного ребенка;
- \forall иметь_ребенка.Женщина – множество людей, все дети которых являются девочками.

Следующие определения применимы не только к логике ALC, но и к любой ДЛ:

- Концепт C называют *выполнимым*, если существует такая интерпретация I , что $C^I \neq \emptyset$. При этом I называют моделью концепта C .
- Концепты C и D называют *эквивалентными* и обозначают $C \equiv D$, если в любой интерпретации I имеем $C^I = D^I$.
- Концепт C называют *вложенным* в концепт D и обозначают $C \sqsubseteq D$, если в любой интерпретации I имеем $C^I \subseteq D^I$.
- Концепты C и D называют *непересекающимися*, если в любой интерпретации I имеем $C^I \cap D^I = \emptyset$.

Очевидно, что справедливы эквивалентности и вложения концептов, являющиеся аналогами законов, выполняющихся для пересечения,

объединения и дополнения множеств. Концепт $\exists R. \perp$ в любой интерпретации обозначает множество элементов, у которых существует R -последователь, принадлежащий пустому множеству $\perp^I = \emptyset$. 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. С другой стороны, \perp тоже всегда интерпретируется как пустое множество. Отсюда следует, что имеет место эквивалентность: $\exists R. \perp \equiv \perp$. Аналогично, $\forall R. \top \equiv \top$.

На практике обычно возникают задачи по заданному концепту выяснить его выполнимость, а также по заданным двум концептам выяснить, имеет ли место их эквивалентность, вложение или непересекаемость. Однако нет необходимости создавать независимые алгоритмы для решения этих задач, так как все они сводятся друг к другу, поскольку для любых концептов C и D справедливо:

- C невыполним $\Leftrightarrow C \sqsubseteq \perp$;
- $C \equiv D \Leftrightarrow C \sqsubseteq D \wedge D \sqsubseteq C$;
- C и D не пересекаются $\Leftrightarrow C \sqcap D \sqsubseteq \perp$.

С другой стороны, очевидно, что для любых концептов C и D справедливо:

- $C \sqsubseteq D \Leftrightarrow$ концепт $C \sqcap \neg D$ невыполним;
- $C \equiv D \Leftrightarrow$ концепты $C \sqcap \neg D$ и $D \sqcap \neg C$ оба невыполнимы;
- C и D не пересекаются \Leftrightarrow концепт $C \sqcap D$ невыполним.

3.3. Терминология логики ALC

Аксиомой называется выражение вида $C \sqsubseteq D$ или $C \equiv D$, где C и D – произвольные концепты. *Терминологией* (или *TBox*) называется произвольный конечный набор аксиом данного вида.

Следующая совокупность аксиом является примером терминологии:

$$\text{Человек} \equiv \text{Мужчина} \sqcup \text{Женщина}$$

Учитель \sqsubseteq Человек

Мать \sqsubseteq Человек $\sqcap \exists \text{ иметь_ребенка}.\top$

Интуитивно эти аксиомы говорят, что всякий человек является мужчиной или женщиной; быть матерью означает быть человеком и иметь ребенка.

Семантика терминологии определяется естественным образом. Пусть дана интерпретация I :

- Аксиома $C \sqsubseteq D$ истинна в интерпретации I , если $C^I \subseteq D^I$; при этом интерпретацию I называют моделью данной аксиомы и пишут $I \models C \sqsubseteq D$.
- Аксиома $C \equiv D$ истинна в интерпретации I , если $C^I = D^I$; при этом интерпретацию I называют моделью данной аксиомы и пишут $I \models C \equiv D$.
- Интерпретацию I называют моделью терминологии T и пишут $I \models T$, если I является моделью всех аксиом из T .
- Терминология $TBox$ называется выполнимой, если она имеет модель.

Пусть $TBox$ – произвольная терминология, C и D – произвольные концепты. Тогда:

- Концепт C выполним в терминологии $TBox$, если существует модель I терминологии $TBox$, такая что $C^I \neq \emptyset$.
- Концепты C и D эквивалентны в терминологии $TBox$ (обозначение: $TBox \models C \equiv D$), если в любой модели I терминологии $TBox$ имеем $C^I = D^I$.
- Концепт C вложен в D в терминологии $TBox$ (обозначение: $TBox \models C \sqsubseteq D$), если в любой модели I терминологии $TBox$ имеем $C^I \subseteq D^I$.
- Концепты C и D называют непересекающимися в $TBox$, если в любой модели I терминологии $TBox$ имеем $C^I \cap D^I = \emptyset$.
- Аксиома α следует из $TBox$ (обозначение: $TBox \models \alpha$), если α истинна в любой модели терминологии $TBox$.

- Две терминологии $TBox_1$ и $TBox_2$ называют эквивалентными (обозначение: $TBox_1 \equiv TBox_2$), если у них одни и те же модели, т.е. для любой интерпретации I она является моделью $TBox_1$ тогда и только тогда, когда она является моделью $TBox_2$.

Пусть I – терминология из предыдущего примера. Тогда концепт Женщина $\Pi \neg \text{Человек}$ невыполним в I , т.к. ввиду первой аксиомы имеем: $I \models \text{Женщина} \sqsubseteq \text{Человек}$. В то же время концепт Мужчина $\Pi \text{Женщина}$ выполним в I . Чтобы этот концепт стал невыполнимым в I , нужно добавить в $TBox$ дополнительные аксиомы.

На практике эксперты в какой-либо предметной области создают терминологию, в которой в виде аксиом фиксируют взаимосвязи основных понятий (концептов и ролей), имеющихся в данной области знаний. После того, как такая система аксиом сформулирована, возникают задачи вывода новых (или, как говорят, неявных) знаний из знаний, заданных явно в терминологии. Одной из первых задач является проверка того, что терминология вообще имеет хотя бы одну модель (т.е. совместна). Далее проверяют, что все атомарные концепты являются выполнимыми в данной терминологии. Если какой-то атомарный концепт оказывается невыполнимым, это обычно является признаком того, что в аксиомах терминологии допущены ошибки. В дальнейшем терминология используется для вывода новых включений и эквивалентностей концептов из уже имеющихся. Соответствующие задачи сводимы друг к другу. Для любых концептов C и D и терминологии $TBox$ справедливы следующие эквивалентности:

- C невыполним в $TBox \Leftrightarrow TBox \models C \sqsubseteq \perp$;
- $TBox \models C \equiv D \Leftrightarrow TBox \models C \sqsubseteq D$ и $TBox \models D \sqsubseteq C$;
- C и D не пересекаются в $TBox \Leftrightarrow TBox \models (C \Pi D) \sqsubseteq \perp$.

Для любых концептов C и D и терминологии $TBox$ справедливы следующие эквивалентности:

- $TBox \models C \sqsubseteq D \Leftrightarrow$ концепт $C \sqcap \neg D$ невыполним в $TBox$;
- $TBox \models C \equiv D \Leftrightarrow$ концепты $C \sqcap \neg D$ и $D \sqcap \neg C$ оба невыполнимы в $TBox$;
- C и D не пересекаются в $TBox \Leftrightarrow$ концепт $C \sqcap D$ невыполним в $TBox$.

3.4. Система фактов логики ALC

Терминологии позволяют записывать общие знания о концептах и ролях. Однако обычно требуется также записать знания о конкретных экземплярах: к какому классу (концепту) они принадлежат, какими отношениями (ролями) они связаны друг с другом. Это делается в той части базы знаний, которая называется системой фактов об индивидах или $ABox$ (от англ. assertional box). С этой целью, помимо множества AC атомарных концептов и множества AR атомарных ролей, т.е. имен для классов и отношений, вводится также конечное множество IN имён экземпляров. Факты об экземплярах бывают двух видов:

- утверждение о принадлежности индивида a концепту C – записывается как $a:C$;
- утверждение о связи двух индивидов a и b ролью R – записывается как aRb .

Определим синтаксис $ABox$. *Системой фактов* называется конечное множество $ABox$ утверждений вида $a: C$ и aRb , где $a, b \in IN$ есть индивиды, C – произвольный концепт, R – роль.

Чтобы задать семантику для $ABox$, надо взять произвольную интерпретацию $I = (\Delta, \cdot^I)$ и расширить ее на экземпляры: каждому экземпляру $a \in IN$ сопоставить элемент области интерпретации $a^I \in \Delta$.

Положим, что факт $a:C$ или aRb *верен* в интерпретации I , если $a^I \in C^I$ или $\langle a^I, b^I \rangle \in R^I$ соответственно. При этом I называется *моделью* этого факта, что записывается как $I \models a:C$ и $I \models aRb$ соответственно.

Интерпретация I называется *моделью* системы фактов $ABox$, если I является моделью всех фактов из $ABox$. $ABox$ называется *выполнимой* (в терминологии $TBox$), если $ABox$ имеет модель (являющуюся одновременно моделью терминологии $TBox$).

Следующая совокупность является системой фактов в языке логики ALC :

Мария: Женщина $\sqcap \neg$ Учитель
 Мария иметь_ребенка Александр
 Александр: Учитель $\sqcap \forall$ иметь_ребенка. \perp

Здесь Мария и Александр есть имена экземпляров. Интуитивно (другими словами, при «естественной» семантике) эти утверждения означают, что Мария является женщиной, но не учителем, у нее есть ребенок Александр, причем Александр является учителем и не имеет детей.

Иногда дополнительно требуют, чтобы семантика $ABox$ удовлетворяла так называемому «*соглашению об уникальности имен*», означающему, что разным именам индивидов интерпретация должна сопоставлять различные элементы из области интерпретации, то есть чтобы отображение $a \in IN \rightarrow a^I \in \Delta$ было инъективным. Это требование вполне естественно; например, именам Мария и Александр из последнего примера логично было бы сопоставлять разные элементы при интерпретации. Однако в дальнейшем по умолчанию не будем требовать выполнения этого соглашения; если же оно будет подразумеваться, то будем говорить об этом явно.

3.5. База знаний логики ALC

База знаний логики ALC – это пара $B = (TBox, ABox)$, где $TBox$ – произвольная терминология, а $ABox$ – произвольная система фактов. Интерпретация I называют *моделью* базы знаний B и обозначают $I \models B$, если

I является одновременно моделью $TBox$ и $ABox$. База знаний называется *выполнимой* (или *совместной*), если она имеет модель.

Говорят, что факт α (вида $a: C$ или aRb) *следует из* B и пишут $B \models \alpha$, если α выполняется в любой модели I базы знаний B .

Дескрипционные логики строятся таким образом, чтобы базы знаний были *разрешимыми* теориями. Более того, обычно требуется, чтобы по теории (базе знаний) и утверждению можно было эффективно (т.е. алгоритмически) установить, является ли данное утверждение следствием данной теории. Сформулируем основные алгоритмические проблемы, которые обычно решаются на практике.

Если имеется лишь терминология $TBox$, то рассматривают следующие алгоритмические проблемы:

T0: совместность терминологии $TBox$;

T1: выполнимость концепта C в терминологии $TBox$;

T2: вложение концептов $C \sqsubseteq D$ в терминологии $TBox$;

T3: классификация терминологии $TBox$, когда требуется для всех атомарных концептов A, B , встречающихся в $TBox$, проверить вложение $A \sqsubseteq B$ в $TBox$ и в качестве результата выдать так называемую таксономию – частично упорядоченное множество всех атомарных концептов относительно вложения.

Основной проблемой является T1. Остальные проблемы к ней сводятся: T0 сводится к T1, так как совместность терминологии $TBox$ равносильна выполнимости концепта \top в $TBox$; T2 сводится к T1; T3 решается путем многократного решения проблемы T2.

Если же даны не только $TBox$, но и $ABox$, то рассматривают следующие логические проблемы:

B0: выполнимость базы знаний B , другими словами, выполнимость $ABox$ относительно $TBox$;

B1: принадлежность экземпляра a концепту C относительно B , т.е. проверка того, что $B \models a: C$;

B2: выборка экземпляров класса (для заданного концепта C найти $\{a \in IN \mid B \models a : C\}$);

B3: классификация экземпляра (найти все минимальные по вложению атомарные концепты, содержащие заданный индивид a относительно B). Более точно, найти все атомарные концепты A , такие что $B \models a : A$ и не существует атомарного концепта A' , такого что $B \models a : A'$, $TBox \models A' \sqsubseteq A$ и $TBox \not\models A \sqsubseteq A'$.

Из этих проблем в качестве основной обычно выбирают $TBox B0$. Остальные к ней сводятся. Действительно, $B \models a : C$ тогда и только тогда, когда база знаний $(TBox, ABox \cup \{a : \neg C\})$ невыполнима; тем самым $B1$ сведена к $B0$. Проблема $B2$ есть многократно решаемая проблема $B1$. Проблема $B3$ комбинирует в себе проблемы $B1$ и $T2$. Более того, проблема $T1$, а значит и все проблемы из первой группы, тоже сводится к $B0$. Действительно, очевидна эквивалентность: концепт C выполним относительно терминологии тогда и только тогда, когда база знаний $(T, \{a : C\})$ выполнима, где a есть новый (нигде не встречавшийся) индивид. Тем самым достаточно научиться решать проблему $B0$ выполнимости баз знаний.

3.6. Разрешимость дескрипционной логики. Понятие разрешающего алгоритма

Установим разрешимость логических проблем, перечисленных в предыдущем разделе, для логики ALC . Для этого опишем так называемый *табло-алгоритм* (tableau algorithm), который проверяет выполнимость баз знаний в этой логике. Пусть мы хотим проверить, верно ли вложение концептов:

$$\exists R.A \sqcap \neg \forall R.(\neg B \sqcap A) \sqsubseteq \exists R.B.$$

Для этого мы должны проверить на выполнимость концепт

$$C := \exists R.A \sqcap \neg \forall R.(\neg B \sqcap A) \sqcap \neg \exists R.B.$$

Если ответ на этот вопрос будет «да», то ответ на исходный вопрос будет «нет» и наоборот.

Нормализуем концепт C , то есть «пронесем» все отрицания внутрь с тем, чтобы получить эквивалентный концепт, в котором все отрицания стоят лишь перед атомарными концептами. Это всегда 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 нашем случае получим нормализованный концепт:

$$C_0 := \exists R.A \sqcap \exists R.(B \sqcup \neg A) \sqcap \forall R.\neg B.$$

Будем строить модель, в которой этот концепт выполним (непуст). 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 обычной $ABox$. Итак, изначально имеем $ABox := \{x : C_0\}$. Поскольку C_0 есть конъюнкция трех концептов, то x должен принадлежать всем трем, поэтому дописываем в $ABox$ следующие факты:

- (1) $x : \exists R.A,$
- (2) $x : \exists R.(B \sqcup \neg A),$
- (3) $x : \forall R.\neg B.$

Ввиду условия (1) должна существовать точка y , связанная с x отношением R , в которой выполнен концепт A . Поэтому добавляем в $ABox$ следующие факты:

- (4) $xRy,$
- (5) $y : A.$

Аналогично, из условия (2) следует существование точки z , связанной с x отношением R , в которой выполнен концепт $B \sqcup \neg A$. Поэтому добавляем в $ABox$ следующие факты:

- (6) $xRz,$
- (7) $z : B \sqcup \neg A.$

Далее, ввиду условия (3) во всех точках, связанных с x отношением R (в нашем случае это точки y и z), должен быть выполнен концепт $\neg B$. Поэтому добавляем в $ABox$ следующие факты:

(8) $y : \neg B$,

(9) $z : \neg B$.

Остается разобрать условие (7). Так как точка z принадлежит дизъюнкции (объединению) концептов, то она принадлежит одному или другому концепту. Поэтому нужно рассмотреть два случая. Первый случай: $(7')z : B$ – немедленно приводит к противоречию с условием (9), согласно которому $z : \neg B$. Второй же случай: $(7'')z : \neg A$ – никаких явных противоречий не создает.

Итак, доведя все условия до элементарных, пришли к протоколу, в котором никаких явных противоречий нет. Можно показать, что эта $ABox$, а вместе с ним и исходный концепт C_0 , имеет модель. Фактически сама $ABox$ и представляет модель: нужно все созданные в процессе работы алгоритма точки считать элементами области интерпретации, а присутствующие в $ABox$ элементарные факты (вида xRy или $y : A$) считать заданием интерпретации атомарных концептов и ролей:

$$I = (\Delta, \cdot^I), \text{ где } \Delta = \{x, y, z\}, A^I = \{y\}, B^I = \emptyset, R^I = \{\langle x, y \rangle, \langle x, z \rangle\}.$$

В такой модели все факты из $ABox$ будут верными. Следовательно, концепт C выполним, а исходное включение концептов – неверно.

В дескрипционной логике традиционно отсутствуют исчисления как таковые. Вместо этого для заданной логики пытаются построить алгоритм, проверяющий, например, истинность аксиом или выполнимость концептов. В ДЛ традиционно разрабатывают алгоритмы для проверки *выполнимости* концептов. Вложенность концептов $C \sqsubseteq D$ эквивалентна *невыполнимости* концепта C , соответственно, и ответы алгоритма будут противоположны тем, что давались при проверке вложенности концептов. Поэтому для проблемы

выполнимости вышеприведенные понятия «меняются местами», и приходим к следующему определению.

Алгоритм G является *разрешающей процедурой* для проблемы выполнимости концептов относительно терминологий для ДЛ, если выполнены следующие три условия:

Завершаемость: для любых $(C, TBox)$ алгоритм G выдает ответ $G(C, TBox)$ через конечное время;

Корректность: для любых $(C, TBox)$ если концепт C выполним относительно $TBox$, то $G(C, TBox) = 1$;

Полнота: для любых $(C, TBox)$ если $G(C, TBox) = 1$, то концепт C выполним относительно $TBox$.

3.7. Табло-алгоритм для логики ALC

Пусть дан концепт C_0 логики ALC , выполнимость которого требуется выяснить. Без ограничения общности можно считать, что C_0 уже нормализован, т.е. все встречающиеся в нем символы отрицания стоят перед атомарными концептами. Строим начальную систему фактов $ABox_0\{x_0 : C_0\}$. Далее на каждом шаге к текущей системе фактов применяется правило и получается одна или две новых системы фактов. Список правил приведен в таблице 1.

Таблица 1: Правила табло-алгоритма для логики ALC.

Правило	Условия применения	Действие
Π-правило	если 1. $x : (C \sqcap D) \in ABox$ 2. $x : C \notin ABox$ или $x : D \notin ABox$	то $ABox' := ABox \cup \{x : C, x : D\}$
⊓-правило	если 1. $x : (C \sqcup D) \in ABox$ 2. $x : C \notin ABox$ или $x : D \notin ABox$	то $ABox' := ABox \cup \{x : C\}$ $ABox'' := ABox \cup \{x : D\}$
∃ -правило	если 1. $x : \exists R.C \in ABox$ 2. нет такого y , что $xRy \in ABox$ и $y : C \in ABox$	то создать новую точку y и $ABox' := ABox \cup \{xRy, y : C\}$
∀ -правило	если 1. $x : \forall R.C \in ABox$ 2. есть такой y , что $xRy \in ABox$ и $y : C \notin ABox$	то $ABox' := ABox \cup \{y : C\}$

Порядок применения правил произволен. Следует заметить, правила для Π, \exists, \forall из текущей $ABox$ создают одну новую систему фактов $ABox'$, тогда как \exists -правило из $ABox$ создает двеновых системы фактов $ABox'$ и $ABox''$ и далее правила применяются к каждому из них независимо.

Таким образом, из исходной $ABox_0$ путем применения описанных правил будет построено дерево поиска, у которого в корне находится $ABox_0$ и у каждой $ABox$ есть 0, 1 или 2 последователя. Применение правил прекращается, если к очередной $ABox$ не применимо ни одно из правил, либо если в $ABox$ содержится явное противоречие, так что применять дальнейшие правила не имеет смысла. Эти два вида $ABox$ будут листьями дерева поиска, их называют *листовыми ABox*. Введем соответствующие термины:

- $ABox$ называется *противоречивой*, если она содержит факты $x : A$ и $x : \neg A$ для некоторого экземпляра x и атомарного концепта A , либо он содержит факт $x : \perp$ для некоторого экземпляра x ;
- $ABox$ называется *полной непротиворечивой*, если оно не является противоречивым, но ни одно из правил табло-алгоритма к ней не применимо.

Если алгоритм встречает полную непротиворечивую $ABox$, то он выдает ответ 1, в этом случае существует модель и оставшиеся ветви дерева поиска алгоритм уже не обходит. Напротив, если алгоритм встречает противоречивую $ABox$, 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. В этом случае моделей нет. Итак, по построению табло-алгоритм возвращает значение 1 тогда и только тогда, когда хотя бы одна из листовых $ABox$ является полной непротиворечивой и возвращает 0 в противном случае.

Пусть задан концепт C_0 и терминология $TBox$. Требуется проверить, выполним ли C_0 относительно $TBox$. Без ограничения общности будем полагать, что концепт C_0 нормализован (то есть в нем все отрицания стоят лишь перед атомарными концептами), а также терминология $TBox$ состоит из единственной аксиомы $\top \sqsubseteq E$, 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 $C_0^I \neq \emptyset$. Как и раньше, берем начальную систему фактов $ABox_0 := \{x_0 : C_0\}$. Введем ряд определений. Говорят, что 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 легко сформулировать – нужно к каждому правилу из таблицы 1 добавить дополнительное предусловие «точка x – активная», а также ввести новое правило, добавляющее концепт E в каждую точку. Получающаяся система правил приведена в таблице 2.

Таблица 2. Правила табло-алгоритма для логики $ALC+TBox$

Правило	Условия применения	Действие
Π-правило	если 0. точка x – активная 1. $x : (C \sqcap D) \in ABox$ 2. $x : C \notin ABox$ или $x : D \notin ABox$	то $ABox' := ABox \cup \{x : C, x : D\}$
⊎-правило	если 0. точка x – активная 1. $x : (C \sqcup D) \in ABox$ 2. $x : C \notin ABox$ и $x : D \notin ABox$	то $ABox' := ABox \cup \{x : C\}$ $ABox'' := ABox \cup \{x : D\}$
∃-правило	если 0. точка x – активная 1. $x : \exists R.C \in ABox$ 2. нет такого y , что $xRy \in ABox$ и $y : C \in ABox$	то создать новую точку y и $ABox' := ABox \cup \{xRy, y : C\}$
∀-правило	если 0. точка x – активная 1. $x : \forall R.C \in ABox$ 2. нет такого y , что $xRy \in ABox$ и $y : C \notin ABox$	то $ABox' := ABox \cup \{y : C\}$
⊤-правило	если 0. точка x – активная	то $ABox' := ABox \cup \{x : E\}$

1.	$x : E \notin ABox$
----	---------------------

Как видно, сам табло-алгоритм остается прежним, за исключением того, что в строчку, где фигурируют правила Π, \exists, \forall , необходимо добавить упоминание $TBox$ -правила, которое из $ABox$ создает одну новую $ABox'$.

ГЛАВА 4. ЯЗЫКИ ОПИСАНИЯ ОНТОЛОГИЙ

4.1. Виды языков описания онтологий

Ключевым моментом в проектировании онтологий является выбор соответствующего языка спецификации онтологий. Для реализации различных онтологий необходимы языки их представления, обладающие достаточной выразительной мощностью и позволяющие избежать «низкоуровневых» проблем при проектировке онтологии. Выделяются три основные концепции создания языков описания онтологий: машинно-ориентированные языки, универсальные языки, языки веб-онтологий.

Специализированные машинные языки онтологического описания – это традиционные языки спецификации онтологий. По типу применяемой логики их классифицируют на языки фреймово-продукционные, дескриптивной логики и логики первого порядка. Широко распространёнными представителями машинных языков являются KIF (Knowledge Interchange Format), CycL (Cycorp Language), F-Logic.

KIF – универсальный машинно-ориентированный язык для обмена данными в рамках выбранной предметной области. KIF имеет декларативную семантику и логическую всесторонность (т.е. предусматривает выражение произвольных предложений в исчислении предикатов первого порядка). Язык обеспечивает представление знаний, используется для описания объектов, функций и отношений. KIF – это язык продукционного типа, где каждая продукция записывается в виде импликации.

CycL – формальный язык, синтаксис которого базируется на логике первого порядка. CycL различает такие сущности, как экземпляры, классы, предикаты и функции. Словарь CycL состоит из термов. Множество термов можно разделить на константы, неатомарные термы и переменные. Термы используются при составлении значащих выражений CycL, из которых формируются утверждения. Совокупность утверждений составляет базу знаний. Исходный код CycL находится в свободном доступе по лицензии

OpenSuse, однако для работы большинства приложений Cycl необходимо дополнительно лицензировать сервер Apache.

F-Logic – онтологический язык, который базируется на логиках первого порядка, однако классы и свойства в нем представлены как термины, а не как предикаты. Язык создавался для осуществления взаимодействия между онтологиями, построенными на основе предикатов, и онтологиями, построенными на основе F-Logic. Создатели определили интуитивные трансляторы для преобразования знаний из предикатных онтологий в F-Logic онтологии и показали, что такой перевод сохраняет логические связи (preserves entailment) для большого количества онтологических языков.

Общим недостатком специализированных машинных языков онтологического описания является сложность их синтаксиса и необходимость в специальных инструментах для интерпретации. Этого недостатка лишены универсальные языки, которые, сохранив описательную полноту для компьютерных систем, обеспечивают семантическую прозрачность для человека.

К универсальным языкам описания онтологий относится XML (Extensible Markup Language) и HTML (HyperText Markup Language). XML – рекомендованный W3C (World Wide Web Consortium – консорциум Всемирной паутины) язык разметки данных. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык создания машино-читаемых документов с простым формальным синтаксисом, обладающим семантической прозрачностью для комфортного восприятия человеком. Основной сферой применения языка XML является Интернет. HTML – стандартный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML. Язык HTML интерпретируется браузерами и отображается в виде документа в удобной для человека форме. Язык XHTML (Extensible HyperText Markup Language)

является более строгим вариантом HTML, он следует всем ограничениям XML и фактически XHTML можно воспринимать как приложение языка XML к области разметки гипертекста.

Более поздние языки, основанные на Web-стандартах, такие как XML, RDF, RDF Schema, DAML+OIL, OWL созданы специально для обмена онтологиями через Web.

RDF (Resource Description Framework) – язык описания метаданных ресурсов, использующий XML-синтаксис. В RDF используется базовая модель данных «объект – атрибут – значение», позволяющая ему сыграть роль универсального языка описания семантики и связей для различных ресурсов. Ресурсы описываются в виде ориентированного размеченного графа – каждый ресурс может иметь свойства, которые в свою очередь также могут быть ресурсами или их коллекциями. Все словари RDF используют базовую структуру, описывающую классы ресурсов и типы связей между ними. Это позволяет использовать разнородные децентрализованные словари, созданные для машинной обработки по разным принципам и методам. Важной особенностью стандарта является расширяемость: можно задать структуру описания источника, используя и расширяя такие встроенные понятия RDF-схем, как классы, свойства, типы, коллекции. Модель схемы RDF включает наследование классов и свойств.

RDF Schema (RDFS) – это семантическое расширение RDF, которое обеспечивает механизмы описания связанных ресурсов и их связей. Система классов и свойств RDF Schema похожа на систему типов языков объектно-ориентированного программирования (ООП) с некоторыми различиями. Так, описательный язык словаря RDF определяет свойства в терминах того класса ресурсов, к которому эти свойства относятся, в отличие от языков ООП, описывающих класс в терминах свойств своих элементов. RDFS предоставляет хорошие базовые возможности для описания словарей типов предметных областей. Одно из ограничений – невозможность с помощью

RDFS выразить аксиоматические знания, т.е. задать аксиомы и правила вывода, построенные на них.

DAML+OIL – семантический язык разметки веб-ресурсов, который расширяет стандарты RDF и RDF Schema за счет более полных примитивов моделирования. В последнюю версию DAML+OIL включен набор дополнительных конструкций для создания онтологий и разметки информации в легко интерпретируемом машиной виде. Онтология DAML+OIL состоит из:

- заголовков (headers);
- элементов классов (class elements);
- элементов свойств (property elements);
- экземпляров (instances).

OWL (Web Ontology Language) – язык представления онтологий следующего поколения после DAML+OIL. Обладает более богатым набором возможностей чем XML, RDF, RDF Schema и DAML+OIL. В OWL реализован мощный механизм семантического анализа. OWL обеспечивает более полную машинную обработку веб-контента, предоставляя наряду с формальной семантикой дополнительный терминологический словарь.

4.1. Язык OWL и его виды. Связь дескрипционных логик с OWL

Язык онтологий OWL разработан для использования приложениями, которые должны обрабатывать содержимое информации, а не только представлять эту информацию. В семантическом вебе онтология содержит описания классов, свойств и их реализацию в виде экземпляров классов. Формальная семантика OWL описывает, как с помощью такой онтологии сделать логические выводы, т.е. получить факты, которые не представлены в онтологии напрямую, но следуют из ее семантики. Эти выводы могут быть основаны на одном документе или множестве распределенных документов, связанных между собой определенным образом.

Язык OWL предполагает открытость, т.е. описания ресурсов не ограничены единственным файлом или областью видимости. Класс OWL, первоначально определенный в одной онтологии, может быть расширен в других онтологиях, причем расширение не должно отменять первоначальное определение.

Для языка OWL определено следующее пространство имен:

<http://www.w3.org/2002/07/owl#>.

Обычно для этого пространства используется префикс owl.

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. Виды OWL не являются независимыми языками, а построены на принципе расширения возможностей. Имеющиеся в ДЛ понятия концепт, роль, экземпляр 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 Lite имеет наименьшую выразительную мощность из всех, но для решения простых задач его может быть достаточно. OWL Lite обладает важнейшим свойством разрешимостью. Именно разрешимость и относительно невысокая вычислительная сложность является главной причиной использования OWL Lite для создания многочисленных практических онтологий (в медицине, биоинформатике и т.п.).
- OWL DL – соответствует дескрипционной логике SHOIN(D). OWL DL обладает большей выразительной мощью, чем OWL Lite. OWL DL также обладает свойством разрешимости, однако

вычислительная сложность у него выше, чем OWL Lite. Разрешимость достигается, в частности, наложением ограничений на синтаксис языка. Так, в OWL DL классу запрещено быть экземпляром.

- OWL Full – не соответствует какой-либо ДЛ. Преимущество языка OWL Full заключается в том, что он базируется на семантике языка RDF, что позволяет говорить о полной как структурной, так и семантической совместимости с RDF: любой правильный документ на языке RDF является правильным документом на языке OWL2 Full, а любой правильный вывод на языке RDF Schema является правильным выводом и на языке OWL2 Full. Недостаток языка OWL Full заключается в том, что его большая выразительная мощность не гарантирует полной (эффективной) поддержки логического вывода

Каждый из этих диалектов (кроме OWL Lite) является расширением предыдущего. Как следствие, любая OWL Lite онтология является OWL DL онтологией, а любая OWL DL онтология является OWL Full онтологией.

4.2. Структура документа OWL

Документ OWL – это документ на языке RDF/XML, который может содержать заголовок OWL, а также содержит определения классов, свойств и сведений о представителях классов. Представители классов (*individuals*) по терминологии OWL – это экземпляры классов.

В качестве расширения файла с документом OWL можно использовать расширения .owl или .rdf. Класс owl:Ontology используется для описания заголовка OWL, который в языке RDF/XML имеет следующий синтаксис:

```
<owl:Ontology rdf:about="ресурс">  
...  
</>
```

В атрибуте `rdf:about` задается наименование онтологии или ссылка на онтологию. Если значение атрибута – пустая строка (""), то наименованием онтологии служит базовый URI (IRI) экземпляра класса `owl:Ontology`. Как правило, это URI (IRI) документа, содержащего онтологию. Исключение является случай, когда в элементе `rdf:RDF` задан атрибут `xml:base`, явно устанавливающий базовый URI (IRI) документа.

Классами, определяющими семантические свойства классов OWL, являются `owl:AnnotationProperty` и `owl:OntologyProperty`.

Свойство `owl:imports` является экземпляром класса `owl:OntologyProperty`. Он определяет URI (IRI) документа, импортируемого в данный документ. Импорт документов может быть вложенным.

Свойство `owl:versionInfo` является экземпляром класса `owl:AnnotationProperty`. Обычно его объектом является строка, содержащая сведения о версии данной онтологии.

Свойство `owl:priorVersion` является экземпляром класса `owl:OntologyProperty`. Это свойство содержит ссылку на другую онтологию, которая таким образом рассматривается как предыдущая версия данной онтологии.

Свойство `owl:incompatibleWith` является экземпляром класса `owl:OntologyProperty`. Это свойство содержит ссылку на другую онтологию, которая таким образом рассматривается как предыдущая версия данной онтологии, несовместимая с данной онтологией.

Свойство `owl:backwardCompatibleWith` является экземпляром класса `owl:OntologyProperty`. Это свойство содержит ссылку на другую онтологию, которая таким образом рассматривается как предыдущая версия данной онтологии, совместимая с данной онтологией.

Если в классе `owl:Ontology` задано свойство `owl:backwardCompatibleWith`, то документ OWL может содержать классы `owl:DeprecatedClass` и `owl:DeprecatedProperty`, описывающие классы и свойства предыдущей версии,

которые в данной версии отменены или заменены другими классами и свойствами.

Класс owl:DeprecatedClass является подклассом класса rdfs:Class, а класс owl:DeprecatedProperty – подклассом класса rdf:Property.

4.3. Описание классов OWL

Классы OWL обеспечивают группирование ресурсов со сходными характеристиками. Каждый класс связан с набором представителей класса, называемым *расширением класса*. Представители класса в расширении класса называются *экземплярами класса*. Каждый класс имеет некоторый содержательный смысл, который связан, но не эквивалентен расширению класса, т.е. два класса могут иметь одинаковые расширения, но являться разными классами.

В языках OWL Lite и OWL DL экземпляр класса не может в то же время классом. В языке OWL Full класс может функционировать как экземпляр другого класса.

Классы OWL описываются с помощью описаний класса, которые затем комбинируются в аксиомы класса. В языке OWL определены шесть типов описаний классов:

- с помощью идентификатора класса;
- с помощью перечисления представителей класса;
- с помощью ограничения свойств;
- с помощью пересечения двух и более описаний классов;
- с помощью объединения двух и более описаний классов;
- с помощью дополнения описания класса.

При использовании первого типа определения класс задается с определенным именем. В остальных типах класс задается как пустой узел со свойством rdf:type, чье значение равно owl:Class.

Описание класса с помощью идентификатора в нотации N3 задается следующим образом:

префикс:имя_класса rdf:type owl:Class,

где префикс – это префикс пространство имен, в котором определен новый класс с именем имя-класса.

В RDF/XML класс задается с помощью элемента owl:Class с атрибутом rdf:ID. Ниже приведен пример объявление класса с именем Office:

```
<owl:Class rdf:ID="Office"/>.
```

Класс owl:Class является подклассом класса rdfs:Class. Класс owl:Class в OWL Lite и OWL DL не может одновременно являться экземпляром своего класса. В OWL Full этого ограничения нет, и оба класса считаются эквивалентными.

Два класса с идентификаторами Thing и Nothing в языке OWL являются предопределенными. Расширением класса для owl:Thing является набор всех представителей этого класса, а расширением класса owl:Nothing является пустой набор. Соответственно каждый класс OWL является подклассом owl:Thing, а класс owl:Nothing является подклассом каждого класса.

Описание класса с помощью перечисления выполняется с помощью свойства owl:oneOf. Значением этого свойства является список представителей класса (его экземпляров). Объявить класс по перечислению можно только в языках OWL DL и OWL Full. Приведем пример объявление класса по перечислению OWL, представителями которого являются цвета RGB:

```
<owl:Class>
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Red"/>
    <owl:Thing rdf:about="#Green"/>
    <owl:Thing rdf:about="#Blue"/>
  </owl:oneOf>
</owl:Class>.
```

Под описанием класса с помощью ограничения свойств понимается описание безымянного класса, все представители которого удовлетворяют заданным ограничениям. Ограничения определяются с помощью класса owl:Restriction, являющегося подклассом класса owl:Class. Значение ограничения (ресурс или литерал) задаются с помощью свойства owl:onProperty. Для ограничения может быть задано только одно свойство owl:onProperty. В OWL определены два вида ограничения свойств: ограничения по значению (value constraints) и ограничения по мощности (cardinality constraints).

Ограничения по значению задают диапазон свойства, используемого в описании данного класса. В OWL определены следующие свойства ограничения по значению: owl:allValuesFrom, owl:someValuesFrom, и owl:hasValue. В OWL Lite единственным типом описания класса, допустимым в качестве объекта в свойствах ограничения по значению является имя класса.

Свойство owl:allValuesFrom связывает ограничиваемый класс либо с описанием класса, либо с диапазоном данных. Это свойство используется для описания класса всех представителей, для которых все значения свойства либо описания классов, либо значения данных в заданном диапазоне. Например, описание ограничения класса hasColor, представителями которого являются только значения класса RGBColor (цвета RGB), может быть представлено следующим образом:

```
<owl:Restriction>
    <owl:onProperty rdf:resource="#hasColor"/>
    <owl:allValuesFrom rdf:resource="#RGBColor"/>
</owl:Restriction>.
```

Свойство owl:someValuesFrom связывает ограничиваемый класс либо с описанием класса, либо с диапазоном данных. Это свойство используется для описания класса всех представителей, для которых по крайней мере одно

значение свойства – экземпляр либо описания класса, либо значения данных в заданном диапазоне. Приведем пример описания ограничения класса `hasColor`, среди представителей которого имеется, по крайней мере, один представитель для белого цвета (`whiteColor`):

```
<owl:Restriction>
    <owl:onProperty rdf:resource="#hasColor"/>
    <owl:allValuesFrom rdf:resource="#whiteColor"/>
</owl:Restriction>.
```

Свойство `owl:hasValue` связывает ограничиваемый класс с заданным значением, которое является либо представителем класса, либо значением данных. Это свойство используется для описания класса всех представителей, для которых по крайней мере одно значение свойства – экземпляр либо описания класса, либо значения данных в заданном диапазоне. Ниже приведен пример описания ограничения класса `hasColor`, среди представителей которого имеется ссылка на представитель класса для черного цвета (`blackColor`):

```
<owl:Restriction>
    <owl:onProperty rdf:resource="#hasColor"/>
    <owl:hasValue rdf:resource="#blackColor"/>
</owl:Restriction>.
```

Ограничение по мощности задает количество значений, которое может иметь свойство. В OWL определены следующие свойства ограничения по мощности: `owl:maxCardinality`, `owl:minCardinality` и `owl:Cardinality`. В OWL Lite свойства ограничения по мощности могут иметь только значения "0" и "1".

Свойство `owl:maxCardinality` связывает ограничиваемый класс со значением данных, принадлежащим к пространству значений типа `nonNegativeInteger` схемы XML. Свойство описывает класс всех

представителей, который имеет самое большое N различных значений, где N – значение ограничения мощности. Описание ограничения класса SmallOfficeStaff, который имеет самое большое 10 представителей, можно представить следующим образом:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#SmallOfficeStaff"/>
  <owl:maxCardinality rdf:datatype=
    "&xsd;nonNegativeInteger">10</owl:maxCardinality>
</owl:Restriction>.
```

Свойство owl:minCardinality связывает ограничиваемый класс со значением данных, принадлежащим к пространству значений типа nonNegativeInteger схемы XML. Свойство описывает класс всех представителей, который имеет самое меньшее N различных значений, где N – значение ограничения мощности. Опишем ограничения класса BigOfficeStaff, который имеет самое меньшее 100 представителей, следующим образом:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#SmallOfficeStaff"/>
  <owl:minCardinality rdf:datatype=
    "&xsd;nonNegativeInteger">100</owl:minCardinality>
</owl:Restriction>.
```

Свойство owl:cardinality связывает ограничиваемый класс со значением данных, принадлежащим к пространству значений типа nonNegativeInteger схемы XML. Свойство описывает класс всех представителей, который имеет точно N различных значений, где N – значение ограничения мощности. Например, ограничения класса SomeOfficeStaff, который имеет точно 8 представителей, может быть описано так:

```

<owl:Restriction>
    <owl:onProperty rdf:resource="#SmallOfficeStaff"/>
    <owl:cardinality rdf:datatype=
        "&xsd;nonNegativeInteger">8</owl:cardinality>
</owl:Restriction>.

```

Описание класса с помощью пересечения (операция AND – логическое И) выполняется с помощью свойства owl:intersectionOf, связывающего класс со списком описаний классов. Это свойство описывает безымянный класс, в котором расширение содержит только те представители, которые содержатся во всех расширениях в описании классов в списке. В OWL Lite значения свойство owl:intersectionOf могут быть только идентификаторами класса и/или ограничениями свойств. Ниже приведен пример описания ограничений, являющихся пересечением двух коллекций цветов, для класса:

```

<owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class>
            <!-- Коллекция из двух элементов:
                черного и красного цвета -->
            <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="#blackColor"/>
                <owl:Thing rdf:about="#redColor"/>
            </owl:oneOf>
        </owl:Class>
        <owl:Class>
            <!-- Коллекция из четырех элементов:
                красного, зеленого, синего
                и белого цветов -->
            <owl:oneOf rdf:parseType="Collection">
                <owl:Thing rdf:about="#redColor"/>
                <owl:Thing rdf:about="#greenColor"/>
                <owl:Thing rdf:about="#blueColor"/>

```

```

<owl:Thing rdf:about="#whiteColor"/>
</owl:oneOf>
</owl:Class>
</owl:intersectionOf>
</owl:Class>.

```

Этот класс будет иметь только одного представителя – redColor (красный цвет), поскольку только этот представитель содержится в обоих списках.

Описание класса с помощью объединения (операция OR – логическое ИЛИ) выполняется с помощью свойства owl:unionOf, связывающего класс со списком описаний классов. Это свойство описывает безымянный класс, в котором расширение содержит все представители, которые содержатся хотя бы в одном из расширений класса в описании классов в списке. Свойство owl:unionOf можно использовать только в OWL DL и OWL Full. Приведем пример описание ограничений, являющихся объединением двух коллекций цветов, для класса:

```

<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class>
      <!-- Коллекция из двух элементов:
           черного и красного цвета -->
      <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#blackColor"/>
        <owl:Thing rdf:about="#redColor"/>
      </owl:oneOf>
    </owl:Class>
    <owl:Class>
      <!-- Коллекция из четырех элементов:
           красного, зеленого, синего
           и белого цветов -->
      <owl:oneOf rdf:parseType="Collection">

```

```

<owl:Thing rdf:about="#redColor"/>
<owl:Thing rdf:about="#greenColor"/>
<owl:Thing rdf:about="#blueColor"/>
<owl:Thing rdf:about="#whiteColor"/>
</owl:oneOf>
</owl:Class>
</owl:unionOf>
</owl:Class>.

```

Этот класс содержит пять представителей: blackColor (черный цвет), redColor (красный цвет), greenColor (зеленый цвет), blueColor (синий цвет) и whiteColor (белый цвет), т.е. представителями класса является сумма представителей обоих списков (представитель redColor есть в обоих списках, но в суммарном списке он будет представлен только один раз).

Описание класса с помощью дополнения (операция NOT – логическое НЕ) выполняется с помощью свойства owl:complementOf, описывающего класс, для которого содержит только те представители, которые не принадлежат к расширению класса, являющегося объектом предложения. Свойство owl:complementOf можно использовать только в OWL DL и OWL Full. Опишем ограничение, являющееся дополнением для класса:

```

<owl:Class>
<owl:complementOf>
<owl:Class rdf:about="#greyColor"/>
</owl:complementOf>
</owl:Class>.

```

Расширение этого класса содержит всех тех представителей, которые не принадлежат классу greyColor.

4.4. Аксиомы классов

Описания классов образуют компоненты для определения классов с помощью аксиом классов. Простейшей формой аксиомы класса является описание класса с помощью идентификатора, однако обычно аксиомы содержат дополнительные компоненты, задающие необходимые и/или достаточные характеристики классов. Для комбинирования описания класса в аксиому класса используются следующие свойства: rdfs:subClassOf, owl:equivalentClass и owl:disjointWith. Свойство rdfs:subClassOf задает, что расширение класса в описании класса является подмножеством расширения класса в описании другого класса:

```
<owl:Class rdf:ID="RGBColor">
    <!-- Класс RGBColor является подклассом
        класса Color -->
    <rdfs:subClassOf rdf:resource="#Color"/>
        <!-- Ограничения на свойство hasValue
            класса RGBColor: один из трех цветов:
            красный, зеленый или синий -->
    <rdfs:subClassOf>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#hasValue"/>
            <owl:someValuesFrom>
                <owl:Class>
                    <owl:oneOf rdf:parseType=
                        "Collection">
                        <owl:Thing rdf:about=
                            "#RedColor"/>
                        <owl:Thing rdf:about=
                            "#GreenColor"/>
                        <owl:Thing rdf:about=
                            "#BlueColor"/>
                    </owl:oneOf>
                </owl:Class>
            </owl:someValuesFrom>
        </owl:Restriction>
    </rdfs:subClassOf>
</owl:Class>
```

```

</owl:someValuesFrom>
</owl:Restriction>
</rdfs:subClassOf>
<!-- Класс RGBColor является дополнением
к классу CMYKColor -->
<rdfs:subClassOf>
<owl:Class>
<owl:complementOf rdf:resource="#CMYKColor"/>
</owl:Class>
</rdfs:subClassOf>
</owl:Class>

```

Свойство `owl:equivalentClass` связывает описание данного класса с описанием другого класса. Использование этого свойства в аксиоме означает, что оба класса имеют одинаковое расширение, т.е. оба расширения класса содержат одинаковый набор представителей. Например:

- Объявление именованного класса `BaseColor` эквивалентным именованному классу `RGBColor`:

```

<owl:Class rdf:about="#BaseColor">
  <owl:equivalentClass rdf:resource="#RGBColor"/>
</owl:Class>

```

- Объявление именованного класса `CMYKColor` эквивалентным безымянному вложенному классу, содержащему расширение из четырех представителей: `CyanColor` (циановый цвет), `MagentaColor` (фиолетовый цвет), `YellowColor` (желтый цвет) и `BlackColor` (черный цвет):

```

<owl:Class rdf:ID="CMYKColor">
  <owl:equivalentClass>
    <owl:Class>

```

```

<owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#CyanColor"/>
    <owl:Thing rdf:about="#MagentaColor"/>
    <owl:Thing rdf:about="#YellowColor"/>
    <owl:Thing rdf:about="#BlackColor"/>
</owl:oneOf>
</owl:Class>
</owl:equivalentClass>
</owl:Class>

```

- Можно задать эквивалентность определяемого класса и вложенного класса в сокращенной форме (без использования элемента owl:equivalentClass). Предыдущий пример при использовании сокращенной формы будет иметь следующий вид:

```

<owl:Class rdf:ID="CMYKColor">
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#CyanColor"/>
        <owl:Thing rdf:about="#MagentaColor"/>
        <owl:Thing rdf:about="#YellowColor"/>
        <owl:Thing rdf:about="#BlackColor"/>
    </owl:oneOf>
</owl:Class>

```

Свойство owl:disjointWith связывает описание данного класса с описанием другого класса. Использование этого свойства в аксиоме означает, что расширения двух классов не имеют общих представителей. Например, объявление об отсутствии общих представителей в классах CMYKColor и RGBColor можно описать так:

```

<owl:Class rdf:about="#CMYKColor">
    <owl:disjointWith rdf:resource="#RGBColor"/>
</owl:Class>

```

4.5. Свойства OWL

В языке OWL определены следующие категории свойств:

- свойства онтологий (ontology properties);
- свойства аннотаций (annotation properties);
- свойства объектов (object properties);
- свойства типизированных данных (datatype properties).

Классами, определяющими свойства онтологий и аннотаций, являются owl:AnnotationProperty и owl:OntologyProperty. Свойства объектов в OWL определяются как экземпляры класса owl:ObjectProperty, а свойства типизированных данных – как экземпляры класса owl:DatatypeProperty. Оба этих класса являются подклассами класса rdf:Property. В языке OWL Full свойства объектов и свойства типизированных данных не являются взаимоисключающими, поскольку значения данных можно рассматривать как представления. Поэтому в OWL Full класс owl:ObjectProperty эквивалентен классу rdf:Property.

Аксиома свойства определяет характеристики свойства. В простейшем случае аксиома свойства просто задает существование свойства, например

```
<owl:ObjectProperty rdf:ID="hasValue"/>
```

Однако чаще аксиомы свойства задают дополнительные характеристики свойств. В языке OWL поддерживаются следующие основные конструкции для характеристик свойств: свойства RDFS; отношения к другим свойствам; глобальные ограничения мощности; характеристики логических свойств.

В OWL DL субъект и объект подсвойства должны быть оба либо свойствами типизированных данных, либо свойствами объектов. В OWL Lite значением свойства rdfs:domain и rdfs:range должен быть только идентификатор класса. Для задания отношения к другим свойствам в языке OWL используются свойства owl:equivalentProperty и owl:inverseOf.

Свойство owl:equivalentProperty используется для указания того, что два свойства имеют одинаковое расширение. Например, объявление об эквивалентности свойств hasColorValue и hasValue можно сделать так:

```
<owl:ObjectProperty rdf:ID="hasColorValue">
    <owl:equivalentProperty rdf:resource="#hasValue"/>
</owl:ObjectProperty>.
```

Свойство имеет направление от домена к диапазону. Иногда необходимо определить отношения в обоих направления. Например, для отношения человек_владеет_автомашиной обратным является отношение автомашина_принадлежит_человеку.

Свойство owl:inverseOf формирует обратное отношение для свойства, например, объявление об том, что у свойства hasDescendant (имеет потомка) существует обратное свойство hasAncestor (имеет предка) описано следующим образом:

```
<owl:ObjectProperty rdf:ID="hasDescendant">
    <owl:inverseOf rdf:resource="#hasAncestor"/>
</owl:ObjectProperty>.
```

Глобальные ограничения мощности задаются с помощью классов owl:FunctionalProperty и owl:InverseFunctionalProperty. Эти ограничения называются глобальными, потому что, к какому бы классу не применялось свойство, ограничения будут выполнены. Класс owl:FunctionalProperty определяет функциональное свойство. Функциональным свойством может быть как свойство объекта, так и свойство типизированного данного. Класс owl:FunctionalProperty является специальным подклассом класса rdf:Property. Приведем пример объявления об том, что свойство объекта fillColor (цвет фона) является функциональным, т.е. для ресурса свойство может принимать одно из значений, определенных в классе RGBColor:

```

<owl:ObjectProperty rdf:ID="fillColor">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Shape"/>
    <rdfs:range rdf:resource="#RGBColor"/>
</owl:ObjectProperty>.

```

Если свойство объявляется обратно функциональным, то объект в утверждении для свойства однозначно определяет субъект (некоторое представление класса). Более точно, если задано утверждение, что P – это обратное функциональное свойство, оно предполагает, что значение y может быть значением P для одного экземпляра x , т.е. не может быть двух различных экземпляров x_1 и x_2 таких, что обе пары (x_1,y) и (x_2,y) являются экземплярами P . Обратное функциональное свойство соответствует понятию ключа в базе данных.

Аксиома обратного функционального свойства задается объявлением свойства как экземпляра класса `owl:InverseFunctionalProperty`, который является подклассом класса `owl:ObjectProperty`. По своему определению обратными функциональными свойствами могут быть только свойства объектов. Поскольку в OWL Full свойства типизированных данных являются подклассами свойств объектов, обратное функциональное свойство может быть определено и для свойств типизированных данных. Такое определение недопустимо в OWL Lite и OWL DL. Например, зададим объявление о том, что каждый объект предложений `birthRegion` (представитель класса `Person`) может однозначно идентифицировать субъект (представитель класса `Region`) :

```

<owl:InverseFunctionalProperty rdf:ID="birthRegion">
    <rdfs:domain rdf:resource="#Region"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:InverseFunctionalProperty>.

```

Характеристики логических свойств (транзитивное или симметричное свойство) задаются с помощью классов owl:TransitiveProperty и owl:SymmetricProperty. Если свойство P задается как транзитивное, это означает, что если пара (x,y) является экземпляром P и пара (y,z) также является экземпляром P , то пара (x,z) – тоже экземпляр P . Аксиома транзитивного свойства задается объявлением свойства как экземпляра класса owl:TransitiveProperty, который является подклассом класса owl:ObjectProperty. В OWL DL для транзитивного свойства нельзя задавать ограничения мощности (ни локальные, ни глобальные) ни для свойств или их суперсвойств, ни для обратных свойств или их суперсвойств. К примеру, объявление транзитивного свойства descendantOf (потомок) можно описать следующим образом:

```
<owl:TransitiveProperty rdf:id="descendantOf">
  <rdfs:domain rdf:resource="#Person"/>
  <rdfs:range rdf:resource="#Person"/>
</owl:TransitiveProperty>.
```

Пусть Иванов_Иван_Петрович, Иванов_Петр_Александрович и Иванов_Александр_Николаевич являются представителями класса Person. Тогда, если для экземпляра Иванов_Петр_Александрович значение свойства descendantOf (потомок) равно Иванов_Иван_Петрович, и для экземпляра Иванов_Александр_Николаевич значение свойства descendantOf (потомок) равно Иванов_Петр_Александрович, то тогда, в силу транзитивности свойства descendantOf при семантической обработке документа можно сделать вывод, что Иванов_Иван_Петрович является потомком экземпляра Иванов_Александр_Николаевич.

Симметричное свойство – это свойство, для которого предполагается, что если пара (x,y) является экземпляром P , тогда пара (y,x) – также экземпляр P . Аксиома симметричного свойства задается объявлением

свойства как экземпляра класса owl:SymmetricProperty, который является подклассом класса owl:ObjectProperty. Для симметричного свойства значения свойств должны быть одинаковы. Например, объявление транзитивного свойства brotherOf (брать) имеет вид:

```
<owl:SymmetricProperty rdf:id="brotherOf">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:SymmetricProperty>
```

Пусть для экземпляра Иванов_Иван_Петрович значение свойства brotherOf равно Иванов_Константин_Петрович. Тогда, в силу симметричности свойства brotherOf при семантической обработке документа можно сделать вывод, что и Иванов_Константин_Петрович является братом экземпляра Иванов_Иван_Петрович.

4.6. Описание экземпляров классов в OWL

Представители классов (экземпляры) в языке OWL определяются с помощью специальных аксиом, также называемых фактами. В OWL для представителей определены два типа фактов:

- факты о принадлежности к классу и значениях свойств;
- факты о тождественности представителей.

Факты о принадлежности к классу и значении свойств задают имя представителя, класс, экземпляром которого является представитель и набор значений свойств представителя. Экземпляр класса не обязательно должен иметь имя, он может быть и безымянным. В этом случае в элементе определения представителя не задается атрибут rdf:id.

Факты о тождественности представителей задают утверждения относительно того, являются ли какие-либо представителя тождественными друг другу, т.е. относятся ли они к одному и тому же ресурсу. В языке OWL определены три конструкции для задания факта тождественности представителей: owl:sameAs, owl:differentFrom и owl:AllDifferent. Свойство

`owl:sameAs` связывает двух представителей отношением тождественности, т.е. ссылки URI (IRI) в них указывают на один и тот же ресурс. Понятие тождественности отличается от понятия эквивалентности следующим образом. Эквивалентность означает, что расширения двух классов одинаковы, но не подразумевает, что эти классы указывают на один и тот же ресурс. В OWL Full класс рассматривается одновременно и как свой представитель, поэтому в нем можно задавать тождественность классов. Таким образом, два класса считаются тождественными, если они реализованы по-разному, но имеют одинаковое содержательное значение. Ниже приведен пример оОбъявление тождественности ссылки на ресурс `Moscow` и ссылки на ресурс `CapitalOfRussia`, поскольку Москва является столицей России:

```
<rdf:Description rdf:about="# Moscow">
    <owl:sameAs rdf:resource="# CapitalOfRussia"/>
</rdf:Description>
```

Свойство `owl:differentFrom` указывает, что два представителя не тождественны друг другу, т.е. ссылки URI (IRI) в них указывают на разные ресурсы. Например, зададим представителя класса с именем `studIvanov1`, отличного от представителей `studIvanov2` и `studIvanov3`:

```
<!-- Задание представителя класса Student
с именем studIvanov1 -->
<inst:Student rdf:ID="studIvanov1">
    <!-- Задание полного имени -->
    <inst:fullName>
        <inst:firstName rdf:datatype="&xsd;token">
            Иван</inst:firstName>
        <inst:surName rdf:datatype="&xsd;token">
            Иванович</inst:surName>
```

```

<inst:secondName rdf:datatype="&xsd;token">
Иванов</inst:secondName>
</inst:fullName>
<!-- Задание даты рождения -->
<inst:birthDate rdf:datatype="&xsd;date">
1991-05-17</birthDate>
<!-- Представитель studIvanov1 отличается
от представителей studIvanov2 и studIvanov3 -->
<owl:differentFrom rdf:resource="#studIvanov2"/>
<owl:differentFrom rdf:resource="#studIvanov3"/>
</inst:Student>.

```

В тех онтологиях, где предполагается уникальность имен, использование свойства `owl:differentFrom` может привести к большому количеству предложений, задающих отличие одного представления от другого. Для таких случаев в OWL введен класс `owl:AllDifferent`, для которого определено свойство `owl:distinctMembers`, связывающее экземпляр `owl:AllDifferent` со списком представителей (это свойство может использоваться только как субъект `owl:AllDifferent`). В этом списке указываются все представители, отличающиеся друг от друга. Зададим всех отличных друг от друга представителей класса `Student`, указанных в примере выше, в одном списке в элементе `owl:AllDifferent`:

```

<owl:AllDifferent>
<owl:distinctMembers rdf:parseType="Collection">
<inst:Student rdf:about="#studIvanov1"/>
<inst:Student rdf:about="#studIvanov2"/>
<inst:Student rdf:about="#studIvanov3"/>
</owl:distinctMembers>
</owl:AllDifferent>.

```

4.5. Профили языка OWL

Спецификация OWL включает в себя ряд так называемых профилей. Одни из них являются известными подмножествами языка OWL DL, а другие обладают большей выразительностью, чем OWL DL, но при этом не поддерживают полную семантику языка OWL Full. Многие существующие онтологии используют только ограниченный набор языковых конструкций, доступных в дескрипционных логиках. Именно это послужило поводом к разработке различных профилей языка OWL. Использование менее выразительного языка позволяет значительно увеличить производительность машин логического вывода. Поэтому стандартная библиотека профилей языка OWL2 с различными вариантами компромиссов между выразительностью и вычислительной сложностью очень полезна на практике.

Для любого профиля справедливо следующее:

- профиль ограничен синтаксисом, причем значение синтаксиса определяется спецификацией языка OWL DL;
- профиль определяется логикой, которая позволяет осуществить вывод за полиномиальное время в зависимости либо от количества фактов в онтологии, либо от размера онтологии в целом.

Рассмотрим профили, определенные в языке OWL, и области их применения.

OWL EL. Профиль EL является расширением дескрипционной логики EL. Его основная особенность – выполнение за полиномиальное время вывода для онтологий с большим количеством аксиом классов. Данный профиль разработан для поддержки выразительной мощности нескольких существующих крупных онтологий в сфере здравоохранения и наук о жизни (например, SNOMED-CT, Gene Ontology и GALEN). Основное преимущество профиля OWL EL заключается в работе с пересечениями

классов и ограничениями существования. Язык считается легким и поддерживает обоснованный и полный вывод за полиномиальное время.

OWL QL. Машины вывода, разработанные для языков OWL DL и OWL EL, оптимизированы для вывода на аксиомах классов и относительно неэффективны для работы с онтологиями, содержащими достаточно простые определения классов и большое количество утверждений об экземплярах. Профиль OWL QL был разработан для обеспечения эффективной обработки запросов именно к таким онтологиям. Профиль использует технологии управления реляционными базами данных. Он основан на семействе дескрипционных логик DL Lite и расширен такими выразительными языковыми примитивами, как аксиомы включения свойств, функциональные и обратно-функциональные объектные свойства.

OWL2 RL. Профиль OWL RL основан на так называемых программах дескрипционной логики (Description Logic Programs, DLP) и обеспечивает взаимодействие между дескрипционными логиками и правилами. Фактически профиль представляет собой крупнейший синтаксический фрагмент языка OWL DL, реализуемый с помощью правил. Это очень важная особенность, так как правила эффективно выполняются параллельно, что позволяет разрабатывать масштабируемые реализации логического вывода. OWL RL отличается от профилей QL и EL тем, что он представляет собой некоторое промежуточное звено между языками OWL DL и OWL Full: Машины вывода, основанные на правилах, могут легко игнорировать ограничения OWL DL (такие как различие между классами и индивидами). Это означает, что реализации правил языка OWL RL можно расценивать как реализации подмножества языка OWL Full. Многие из наиболее масштабируемых машин вывода для семантического веба основаны на использовании языка OWL RL или очень похожего языка, называемого pD* или OWL-Horst. Набор правил, который должен быть

реализован в таких приложениях, опубликован в спецификации профиля OWL RL.

ГЛАВА 5. ОНТОЛОГИЧЕСКИЙ ИНЖИНИРИНГ В СИСТЕМЕ PROTEGE

5.1. Программный инструментарий для онтологического инжиниринга

При проектировании и разработке онтологий целесообразно пользоваться подходящими инструментами. Инструментальная поддержка одновременно важна как для процесса разработки, так и для использования онтологических моделей в различных областях. Будем называть инструментальные программные средства, созданные специально для проектирования, редактирования и анализа онтологий, редакторами онтологий. Основная функция любого редактора онтологий состоит в поддержке процесса формализации знаний и представлении онтологии как спецификации (точного и полного описания). В большинстве своем современные редакторы онтологий предоставляют средства кодирвоания формальной модели в том или ином виде. Некоторые дают дополнительные возможности по анализу онтологии, используют механизм логического вывода. Рассмотрим основные характеристики редакторов онтологий.

Поддерживаемые редактором формализмы и форматы представления. Под формализмом понимается теоретический базис, лежащий в основе способа представления онтологических знаний. Примерами формализмов могут служить логика предикатов (First Order Logic - FOL), дескриптивная логика, фреймовые модели (Frames), концептуальные графы и т.п. Формализм, используемый редактором, может не только существенно влиять на внутренние структуры данных, но и определять формат представления или даже пользовательский интерфейс.

Формат представления онтологии в редакторе. Формат представления онтологии задает вид хранения и способ передачи онтологических описаний. Под форматами подразумеваются языки представления онтологий: RDF, OWL, KIF, SCL. Таким образом, некоторая

формальная модель представляется в формализме FOL и может быть выражена средствами языка KIF. Редакторы онтологий обычно поддерживают работу с несколькими формализмами и форматами представления, но часто только один формализм является «родным» (native) для данного редактора.

Функциональность редактора онтологий. Важной характеристикой является функциональность редактора, т.е. множество сценариев его использования. Базовый набор функций обеспечивает:

- работу с одним или более проектами;
- сохранение проекта в нужном формализме и формате (экспорт);
- открытие проекта;
- импорт из внешнего формата;
- редактирование метаданных проекта (в широком смысле: от настройки форм редактирования и представления данных до поддержки версий проекта);
- редактирование онтологии. Набор возможных действий обычно включает создание, редактирование, удаление понятий, отношений, аксиом и прочих структурных элементов онтологии, редактирование таксономии.

К дополнительным возможностям редакторов относят поддержку языка запросов (для поиска нетривиальных утверждений), анализ целостности, использование механизма логического вывода, поддержку многопользовательского режима, поддержку удаленного доступа через Интернет.

Наличие сложных инструментальных средств. Эти средства нужны для того, чтобы не только вводить и редактировать онтологическую информацию, но и анализировать ее, выполняя типичные операции над онтологиями, например, выравнивание, отображение, объединение онтологий.

Рассмотрим наиболее известные инструменты инженерии онтологий.

Ontolingua. Система Ontolingua была разработана в KSL (Knowledge Systems Laboratory) Стенфордского университета и стала первым инструментом инженерии онтологий. Она состоит из сервера и языка представления знаний. Сервер Ontolingua организован в виде набора онтологий, относящихся к Web-приложениям, которые надстраиваются над системой представления знаний Ontolingua. Редактор онтологий – наиболее важное приложение сервера Ontolingua является Web-приложением на основе форм HTML. Кроме редактора онтологий Сервер Ontolingua включает сетевое приложение Webster (получение определений концептов), сервер ОКВС (доступ к онтологиям Ontolingua по протоколу ОКВС) и Chimaera (анализ, объединение, интегрирование онтологий). Все приложения, кроме сервера ОКВС, реализованы на основе форм HTML. Система представления знаний реализована на Lisp. Сервер Ontolingua также предоставляет архив онтологий, включающий большое количество онтологий различных предметных областей, что позволяет создавать онтологии из уже существующих. Сервер поддерживает совместную разработку онтологии несколькими пользователями, для чего используются понятия пользователей и групп. Система включает графический браузер, позволяющий просмотреть иерархию концептов, включая экземпляры. Ontolingua обеспечивает использование принципа множественного наследования и богатый набор примитивов. Сохраненные на сервере онтологии могут быть преобразованы в различные форматы для использования другими приложениями, а также импортированы из ряда языков в язык Ontolingua.

OntoEdit. Редактор OntoEdit первоначально был разработан в институте AIFB (Institute of Applied Informatics and Formal Description Methods) и обеспечивает проверку, просмотр, кодирование и

модификацию онтологий. В настоящее время OntoEdit поддерживает языки представления: FLogic, включая машину вывода, OIL, расширение RDFS и внутреннюю, основанную на XML, сериализацию модели онтологии используя OXML - язык представления знаний OntoEdit (OntoEdit's XML-based Ontology representation Language). К достоинствам инструмента можно отнести удобство использования; разработку онтологии под руководством методологии и с помощью процесса логического вывода; разработку аксиом; расширяемую структуру посредством плагинов, а также очень хорошую документацию. OntoEdit - автономное Java-приложение, которое можно локально установить на компьютере, но его коды закрыты. Существует две версии OntoEdit: свободно распространяемая OntoEdit Free (ограничена 50 концептами, 50 отношениями и 50 экземплярами) и лицензированная OntoEdit Professional (нет ограничений на размер). Естественно, что OntoEdit Professional имеет более широкий набор функций и возможностей (например, машину вывода, графический инструмент запросов, больше модулей экспорта и импорта, графический редактор правил, поддержка баз данных JDBC и т.д.).

OilEd. OilEd – автономный графический редактор онтологий, разработан в Манчестерском университете в рамках европейского IST проекта On-To-Knowledge. Инструмент основан на языке OIL (сейчас адаптирован для DAML+OIL, в перспективе - OWL), который сочетает в себе фреймовую структуру и выразительность дескриптивной логики (Description Logics) с сервисами рассуждения. Из недостатков можно выделить отсутствие поддержки экземпляров. Существующая версия не обеспечивает полную среду разработки – не поддерживается разработка онтологий большого масштаба, миграция и интеграция онтологий, контроль версий и т.д. OilEd свободно распространяется по общедоступной лицензии GPL.

WebOnto. Редактор *WebOnto* предназначен для поддержки совместного просмотра, создания и редактирования онтологий. Его достоинства – простота использования, предоставление средств масштабирования для построения больших онтологий. Для моделирования онтологий *WebOnto* использует язык OCML (Operational Conceptual Modeling Language). В *WebOnto* пользователь может создавать структуры, включая классы с множественным наследованием, что можно выполнять графически. Все слоты наследуются корректно. Инструмент проверяет вновь вводимые данные контролем целостности кода OCML. Инструмент имеет ряд полезных особенностей: сохранение структурных диаграмм, раздельный просмотр отношений, классов, правил и т.д. Другие возможности включают совместную работу нескольких пользователей над онтологией, использование диаграмм, функций передачи и приёма и др.

KADS22. *KADS22* – инструмент поддержки проектирования моделей знаний согласно методологии CommonKADS. Модели CommonKADS определены в CML (Conceptual Modeling Language). *KADS22* представляет интерактивный графический интерфейс для CML со следующими функциональными возможностями: синтаксический анализ файлов CML, печать, просмотр гипертекста, поиск, генерация глоссария и генерация HTML.

Protégé. *Protégé* – локальная, свободно распространяемая Java-программа, разработанная группой медицинской информатики Стенфордского университета. Программа предназначена для построения (создания, редактирования и просмотра) онтологий прикладной области. Её первоначальная цель – помочь разработчикам программного обеспечения в создании и поддержке явных моделей предметной области и включение этих моделей непосредственно в программный код. *Protégé* включает редактор онтологий, позволяющий проектировать онтологии, разворачивая иерархическую структуру абстрактных или

конкретных классов, свойств. Структура онтологии сделана аналогично иерархической структуре каталога. На основе сформированной онтологии Protégé может генерировать формы получения знаний для введения экземпляров классов и подклассов. Инструмент имеет графический интерфейс, удобный для использования неопытными пользователями, снабжен справками и примерами. Protégé основан на фреймовой модели представления знания OKBC (Open Knowledge Base Connectivity) и снабжен рядом плагинов, что позволяет его адаптировать для редактирования моделей хранимых в разных форматах (стандартный текстовый, в базе данных JDBC, UML, языков XML, XOL, SHOE, RDF и RDFS, DAML+OIL, OWL).

5.1. Создание нового проекта в PROTEGE

Запустите Protégé. После запуска программы появится окно проекта Protégé. Пользовательский интерфейс состоит из главного меню и нескольких вкладок. Новый проект открывается на вкладке «Active Ontology», представленной на рис. 5.1.

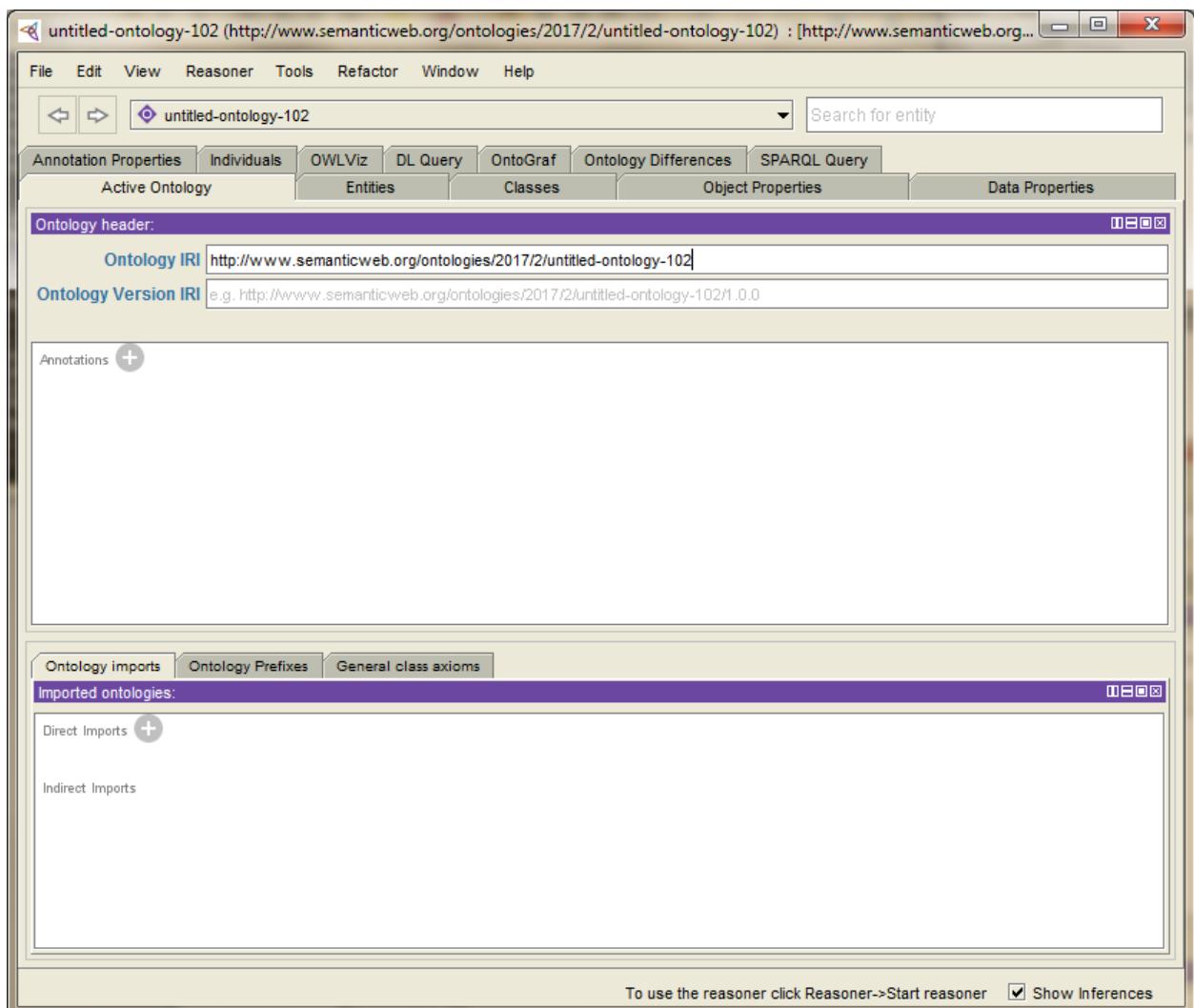


Рисунок 5.1. Вкладка «Active Ontology»

Protégé позволяет работать с несколькими онтологиями сразу. На вкладке «Active Ontology» представлена информация о той онтологии, с которой пользователь работает в настоящий момент, а именно:

- IRI (Internationalized Resource Identifier, международный идентификатор ресурса) текущей онтологии и IRI ее версии соответственно, присваиваемые по умолчанию;
- комментарии пользователя к текущей онтологии.

Создание стандарта IRI имеет целью преодоление ограниченных возможностей стандарта Uniform Resource Identifier (URI). Ограниченностей возможностей идентификации ресурсов, предусмотренных стандартом URI, заключается в том, что такие идентификаторы могут содержать только латинские символы и знаки

препинания из набора символов ASCII. В случае использования символов национальных алфавитов в URI потребуется кодировать их в Unicode, а символы Unicode, в свою очередь, представлять в URI в виде специальных комбинаций символов ASCII. В результате идентификатор URI становится нечитаемым. Использование IRI позволяет применять символы Unicode. IRI определяет возможности использования для идентификации ресурсов Internet удобочитаемых строк литер. Эти строки могут быть, в частности, мнемоничными и содержать литеры национальных алфавитов.

Замените заданный по умолчанию идентификатор онтологии на <http://www.semanticweb.org/ontologies/Lab> (рис. 5.2):

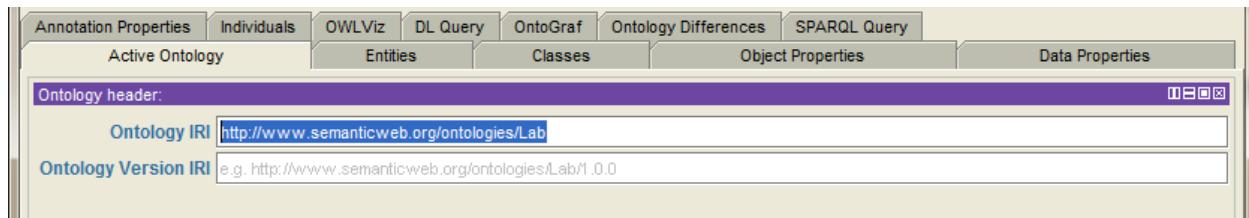


Рисунок 5.2. Изменение IRI онтологии

Вкладка «Active Ontology» имеет несколько вкладок: «Ontology Imports», «Ontology Prefixes», «General class axioms».

Вкладка «Ontology Imports» содержит средства для импорта других онтологий и их слияния с текущей. Импорт другой онтологии переносит весь набор утверждений, обеспеченных в той онтологии, в текущую онтологию.

Вкладка «Ontology Prefixes». Каждому объекту, созданному в Protégé, присваивается международный идентификатор ресурса (IRI), который достаточно длинный. Данная вкладка предназначена для указания аббревиатур (префиксов) для используемых IRI.

Вкладка «General class axioms» предназначена для задания общих (генеральных) аксиом класса.

5.2. Создание комментариев к онтологии

Protégé позволяет аннотировать (пояснять) классы, свойства, экземпляры классов и саму онтологию с помощью фрагментов информации (методанных). Ограничения на использовании аннотации:

- в аннотации должны находиться либо литералы данных (например, «Иван», 18, 3.14), либо ссылка URI или экземпляры классов;
- свойства аннотации не могут быть использованы в аксиомах свойств.

Для создания комментария на вкладке «Active Ontology» щелкните значок (+) рядом с «Annotations». Появится окно редактирования аннотации «Create Annotation» (рис. 5.3).

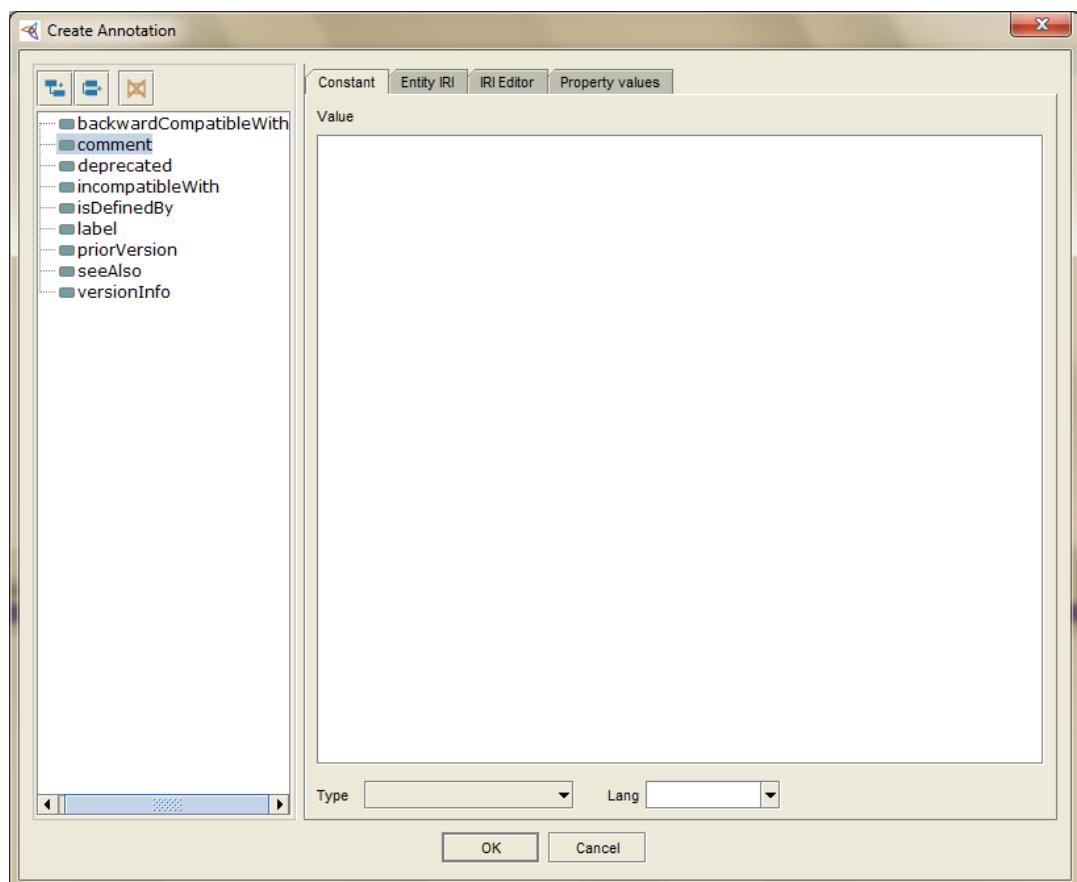


Рисунок 5.3. Окно «Create Annotation»

В Protégé имеется несколько предопределенных свойств аннотации, которые могут быть использованы для аннотирования классов, свойств, экземпляров:

- **comment** – свойство предназначено для пояснения смысла элемента онтологии, имеет диапазон значений «строка»;
- **versionInfo** – значение свойства хранит информацию о версии, диапазон значений атрибута – «строка»;
- **backwardCompatibleWith** – значение свойства определяет предыдущую версию онтологии, совместимую с текущей версией;
- **incompatibleWith** – значение свойства определяет предыдущую версию онтологии, не совместимую с текущей.
- **label** – свойство используется для придания смысла именам элементов онтологии и имеет диапазон значений «строка»;
- **seeAlso** – свойство имеет значение «URI-ссылка» и используется для идентификации соответствующих ресурсов;
- **isDefinedBy** – свойство имеет значение «URI-ссылка» и используется для ссылки на онтологию;
- **priorVersion** – свойство предназначено для идентификации предыдущей версии онтологии;

Выберите из списка в левой части окна пункт «comment» и введите текст «Онтология профессиональной деятельности врача» в текстовое поле в правой части окна.

Нажмите кнопку «OK», чтобы сохранить комментарий. На вкладке «Active Ontology» появится введенный комментарий (рис. 5.4)

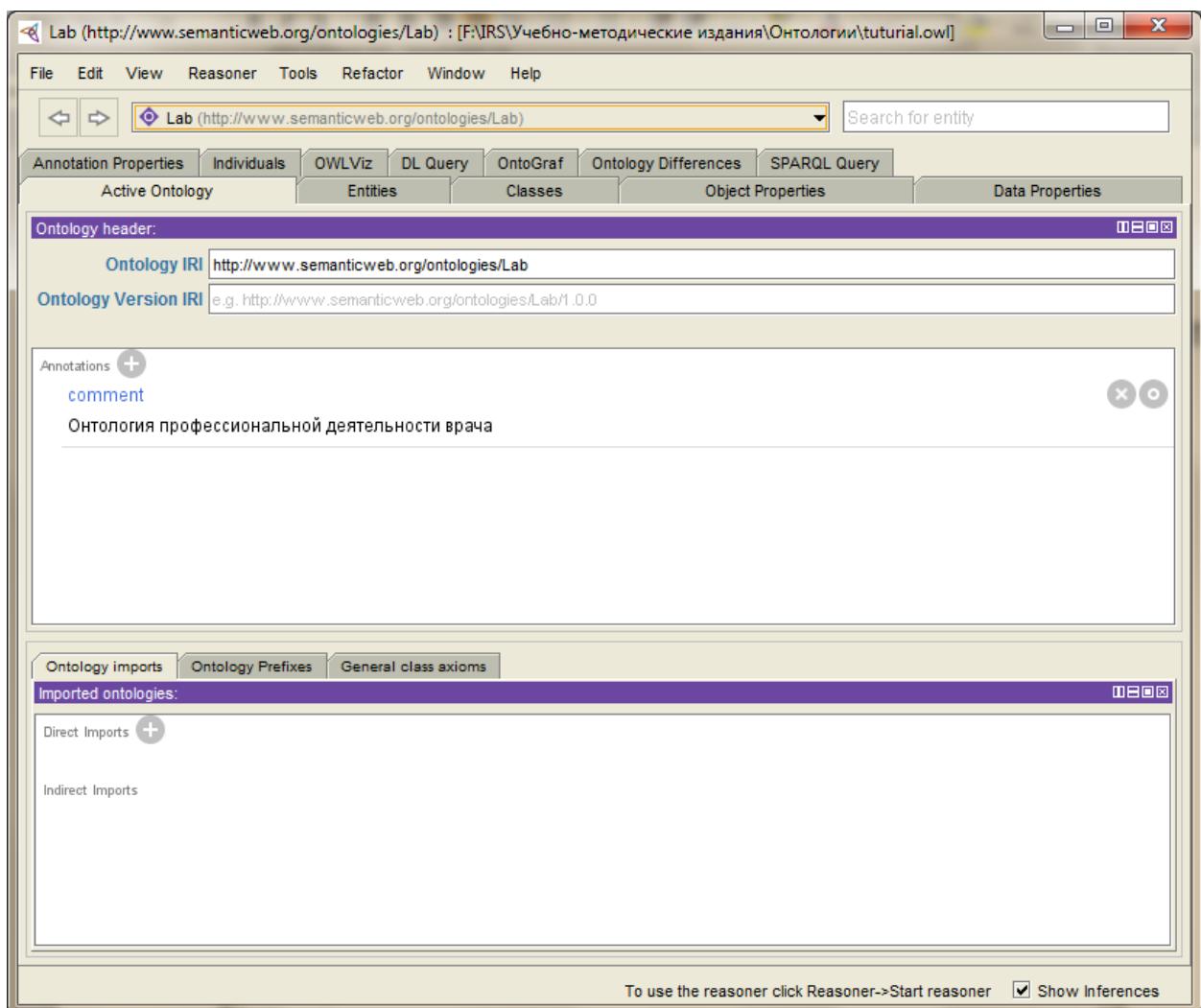


Рисунок 5.4. Вкладка «Active Ontology»

5.3. Сохранение проекта

Для сохранения промежуточных изменений выберите пункт меню «File» ⇒ «Save as». В открывшемся диалоговом окне (рис. 5.5) выберите формат сохранения онтологии «RDF/XML» и нажмите кнопку «OK».

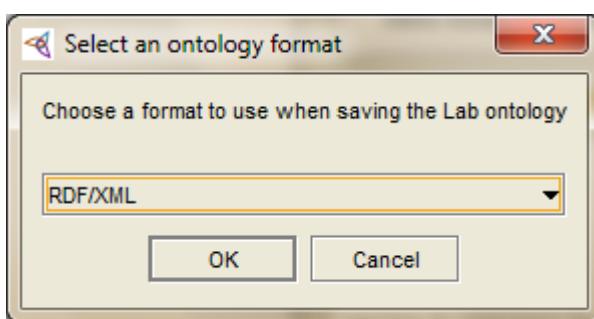


Рисунок 5.5. Окно выбора формата сохранения онтологии

В открывшемся окне (рис. 5.6) укажите место, куда будет сохранен проект, и введите имя файла проекта (например, «tutorial»). Нажмите кнопку **Save**.

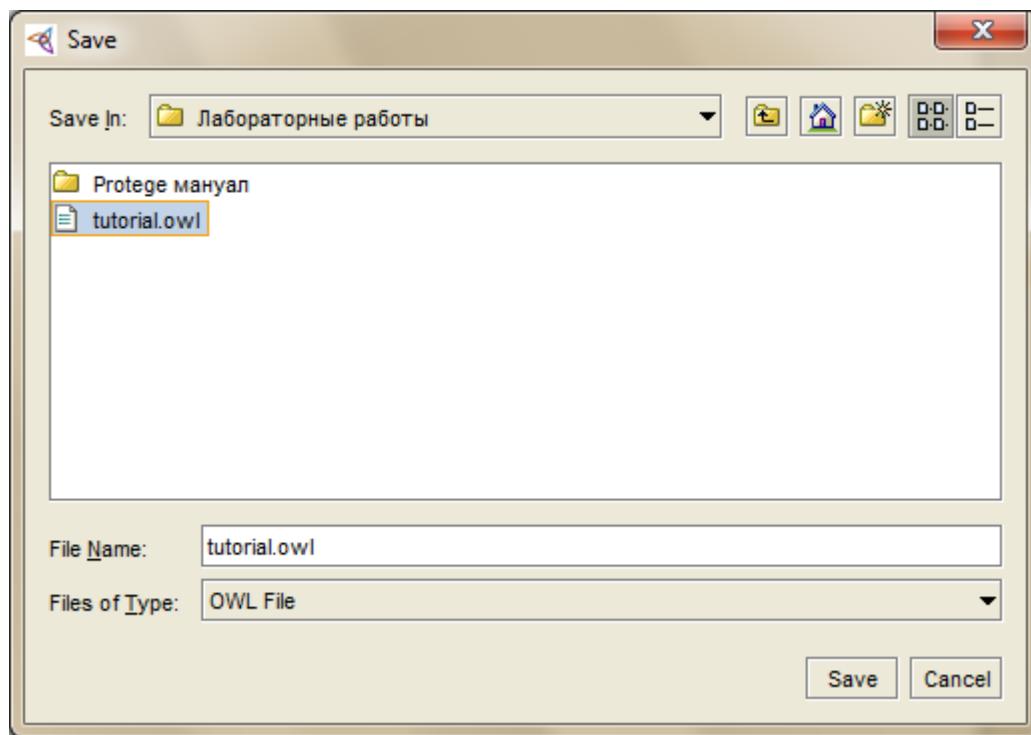


Рисунок 5.6. Окно сохранения проекта

5.4. Способы описания классов

Класс онтологии интерпретируется как некоторое множество объектов предметной области. Иерархия классов описывается логической формулой – импликацией $A \rightarrow B$.

Класс может быть описан несколькими способами:

- идентификатором класса (IRI);
- перечислением всех экземпляров класса;
- ограничением на значение свойства;
- с помощью теоретико-множественных операций (пересечение 2-х и более классов, объединение, дополнением).

Только первый способ определяет именованный класс OWL. Все оставшиеся определяют анонимный (безымянный) класс через ограничение его экстенсионала. Второй способ явно перечисляет

экземпляры класса, третий ограничивает экстенсионал только теми экземплярами, которые удовлетворяют данному свойству. Четвертый способ используют теоретико-множественные операции (объединение, пересечение и дополнение) над экстенсионалами соответствующих классов, чтобы определить экстенсионал нового класса.

Классы образовывают иерархию обобщения (родительский класс – подкласс), иначе называемую таксономией. В Protege существует класс owl:Thing, содержащий все объекты. Данный класс является родительским по отношению ко всем остальным именованным классам. Средства автоматической категоризации на основании онтологии проверяют непротиворечивость иерархии классов и достраивают ее.

5.5. Создание именованного класса

Перейдите на вкладку «Classes» (рис. 5.7). На вкладке «Classes» найдите область «Class Hierarchy» (в окне Protégé слева). Эта область отображает иерархию классов с выделенным текущим выбранным классом. Пустая онтология содержит один класс с именем **THING**. Класс **THING** – это универсальное множество, содержащее все объекты предметной области. По этой причине все классы являются подклассами **THING**.

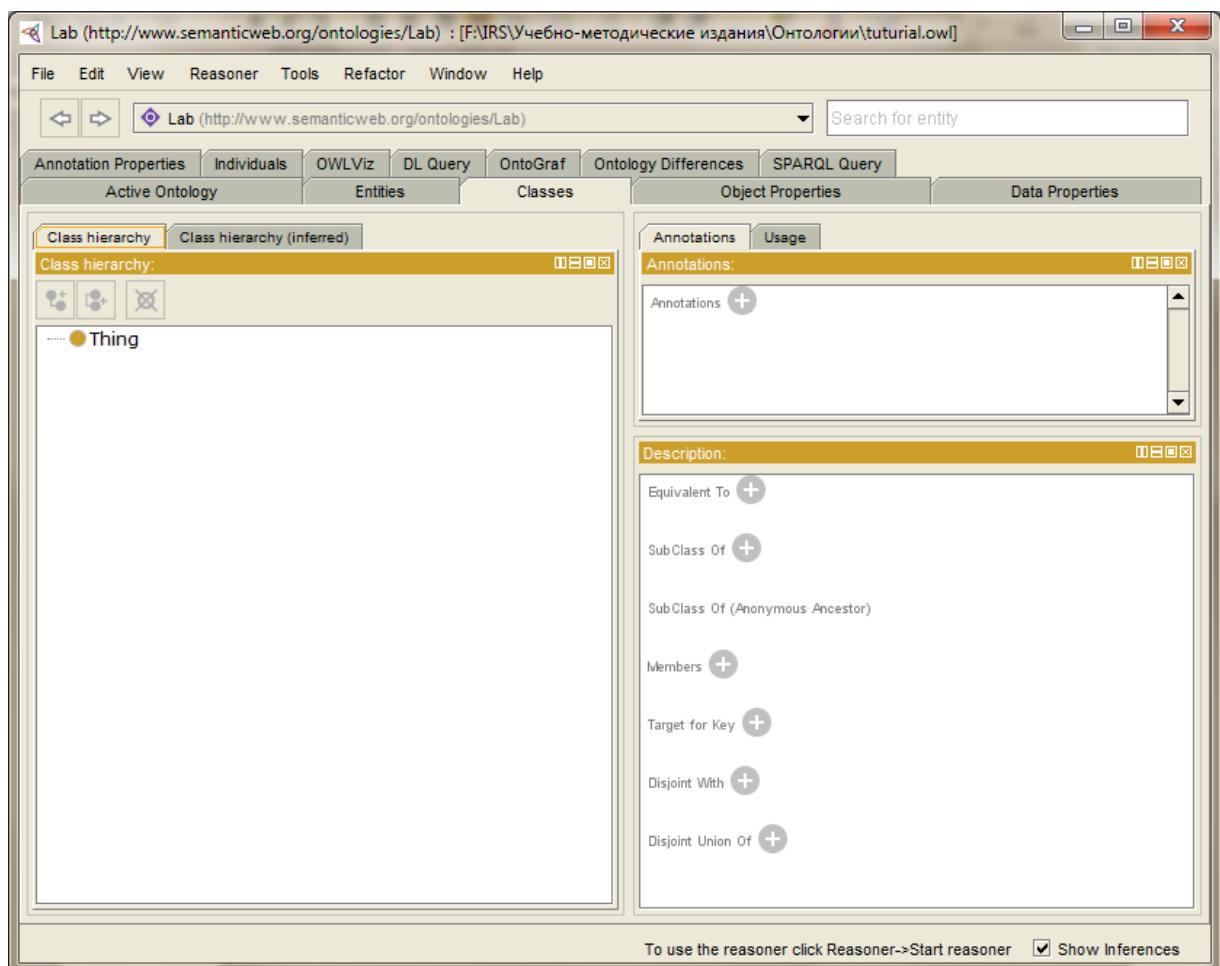


Рисунок 5.7. Вкладка «Classes»

Выделите класс **THING** и нажмите кнопку – «Add subclass» («Создать подкласс») (рис. 7)

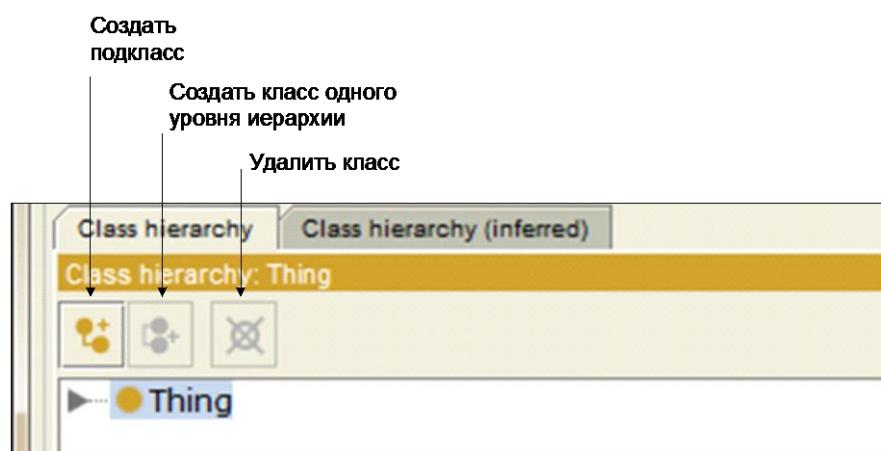


Рис. 5.8. Создание подкласса

Появится окно редактора классов (рис. 5.9). В системе Protégé приняты правила наименования: все имена классов должны начинаться с

прописной буквы и не содержать пробелов. Введите имя класса – Врач и нажмите кнопку «OK».

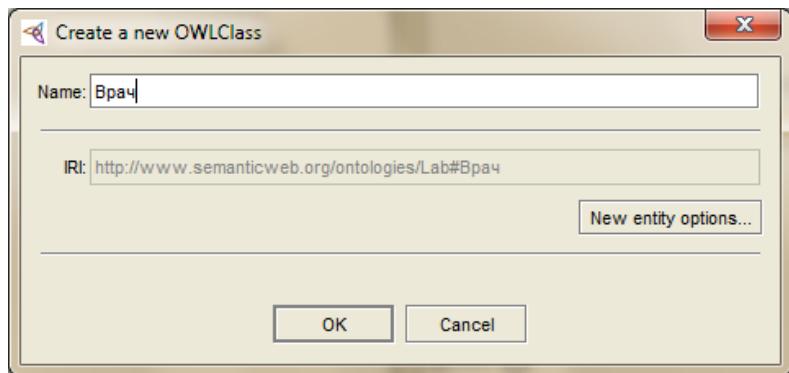


Рисунок 5.9. Окно создания нового класса

Дерево иерархии классов примет вид, представленный на рис. 5.10. Переименовать созданный класс можно с помощью команды меню «Refractor» ⇒ «Rename entity».

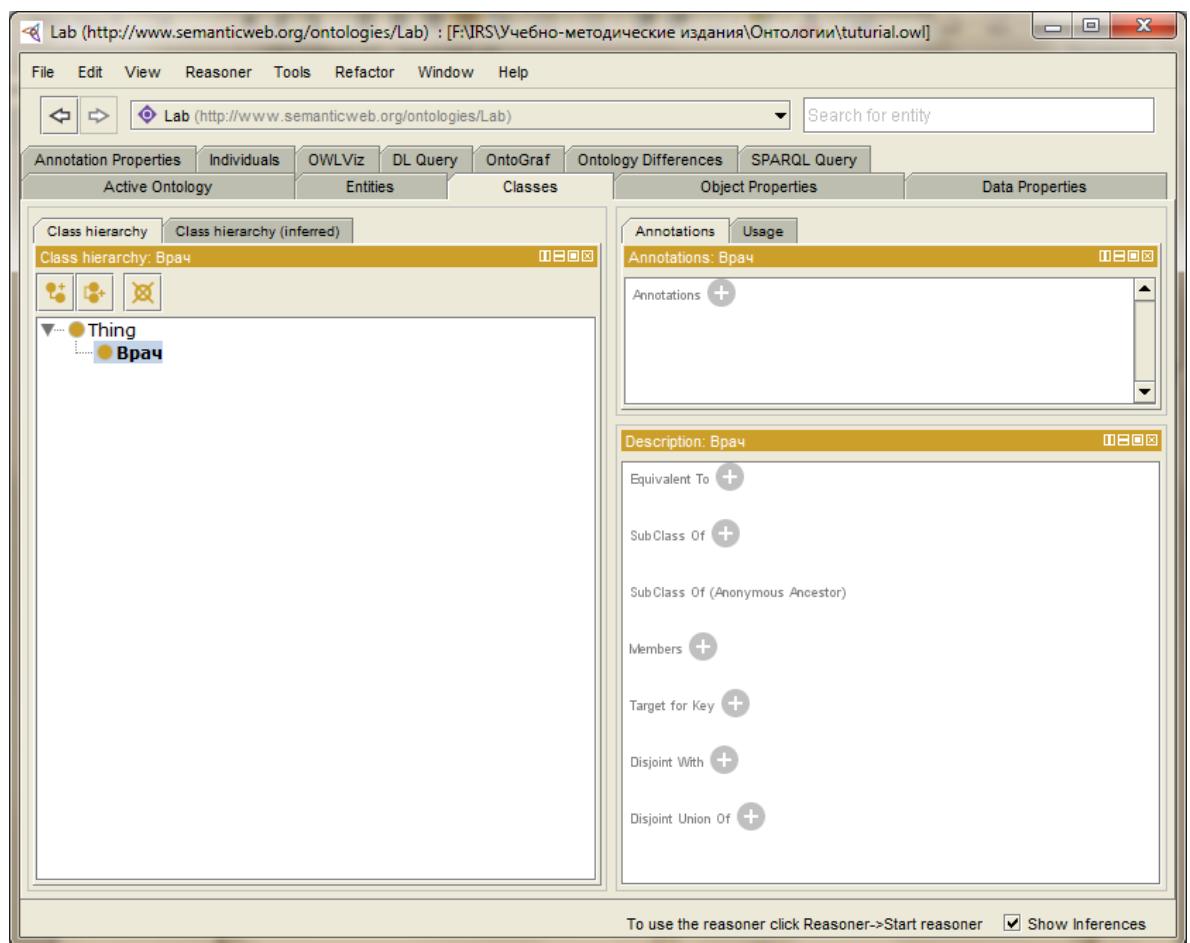


Рисунок 5.10. Иерархия классов

На вкладке «Classes» найдите вкладку «Annotations» (в окне Protégé справа). Выделите класс Врач и щелкните значок (+) рядом с

«Annotations». Появится окно редактирования аннотации к классу Врач. Выберите из списка в левой части окна пункт «comment» и введите текст «Специалист с законченным высшим медицинским образованием» в текстовое поле в правой части окна. Нажмите кнопку «OK», чтобы сохранить комментарий. На экране отобразится созданный комментарий (рис. 11)

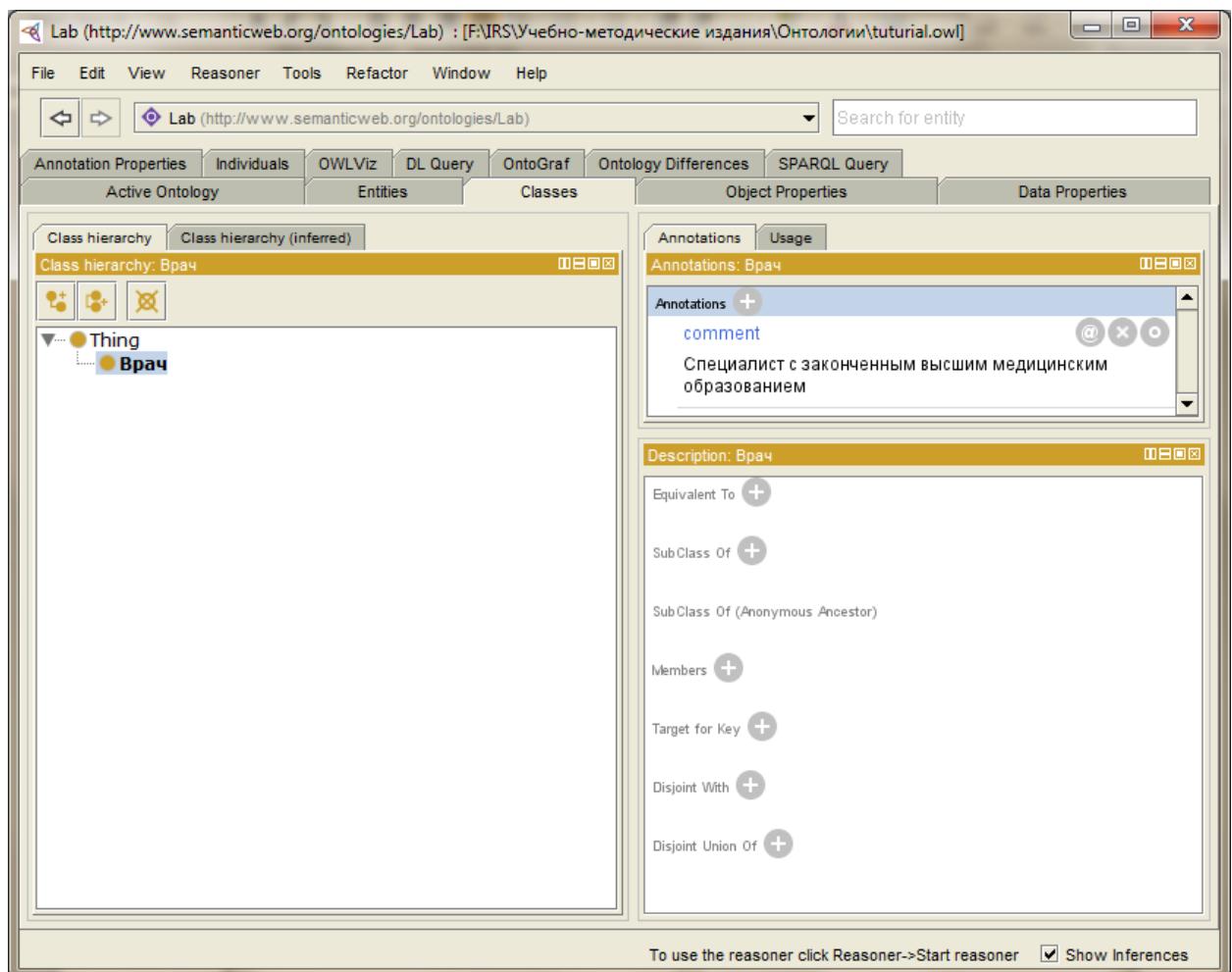


Рисунок 5.11. Создание комментария к классу

Создадим класс Пациент. Для создания нового класса выделите класс THING. Если этого не сделать, то новый класс будет подклассом класса Врач. Нажмите кнопку «Add subclass» . В открывшемся окне наберите с клавиатуры имя класса: Пациент. Нажмите кнопку «OK» для завершения создания класса.

В представлении «Annotations: Пациент» добавьте комментарий к классу Пациент: «Физическое лицо, обратившееся за медицинской помощью, находящееся под медицинским наблюдением либо получающее медицинскую помощь». На экране отобразится созданный комментарий (рис. 12).

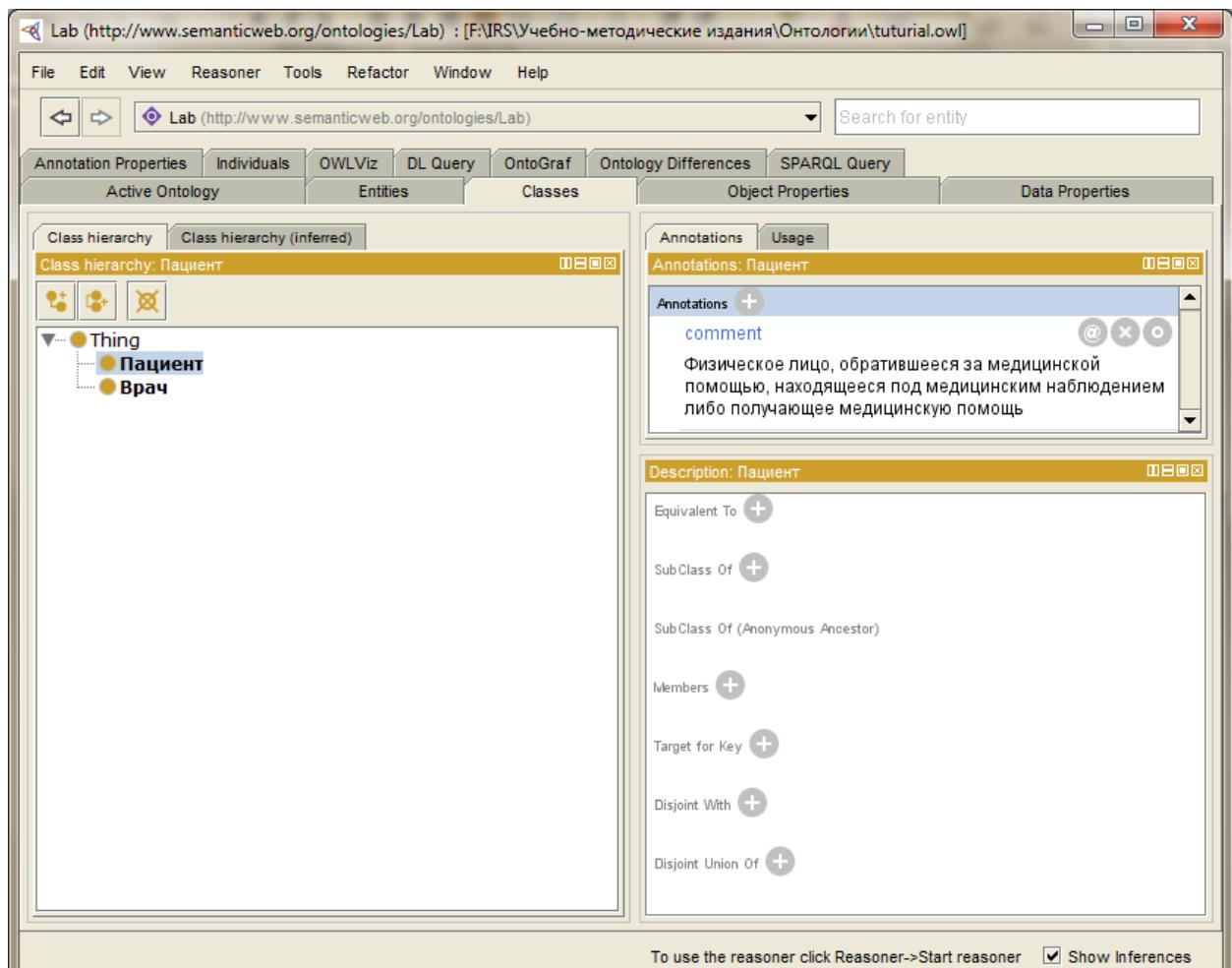


Рисунок 12. Создание класса Пациент

Для создания иерархии классов также можно использовать кнопку – «Add sibling class» («Создать класс одного уровня иерархии»). При нажатии на эту кнопку будет создан класс, находящийся в дереве иерархии на одном уровне с выделенным классом.

Выделите класс Врач и нажмите кнопку – «Add sibling class». В окне редактирования классов введите: Диагноз. Нажмите кнопку «OK».

Ведите аннотацию: «Заключение о сущности болезни и состоянии пациента, выраженное в принятой медицинской терминологии и основанное на всестороннем систематическом изучении пациента». Иерархия классов примет вид, представленный на рис. 13.

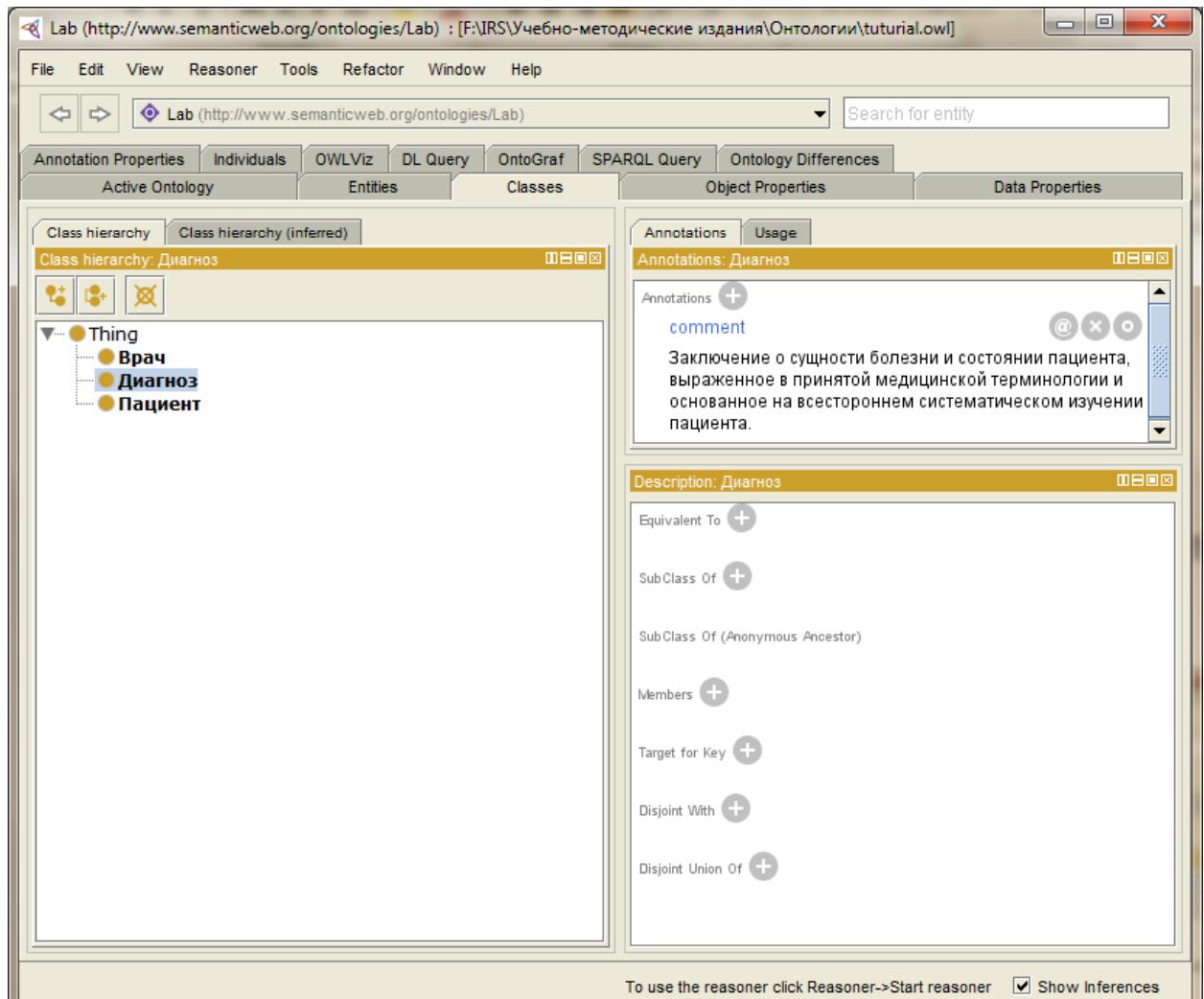


Рисунок 13. Создание класса Диагноз

По умолчанию классы онтологии могут пересекаться. Для того чтобы разделить классы, их необходимо явно сделать непересекающимися.

Объявим, что классы Врач и Диагноз не пересекаются. Для этого выделите класс Врач в иерархии классов и щелкните значок (+) рядом с пунктом «Disjoint with» («Не пересекается с»), который расположен в нижней части области описания класса «Description» (рис. 5.14).

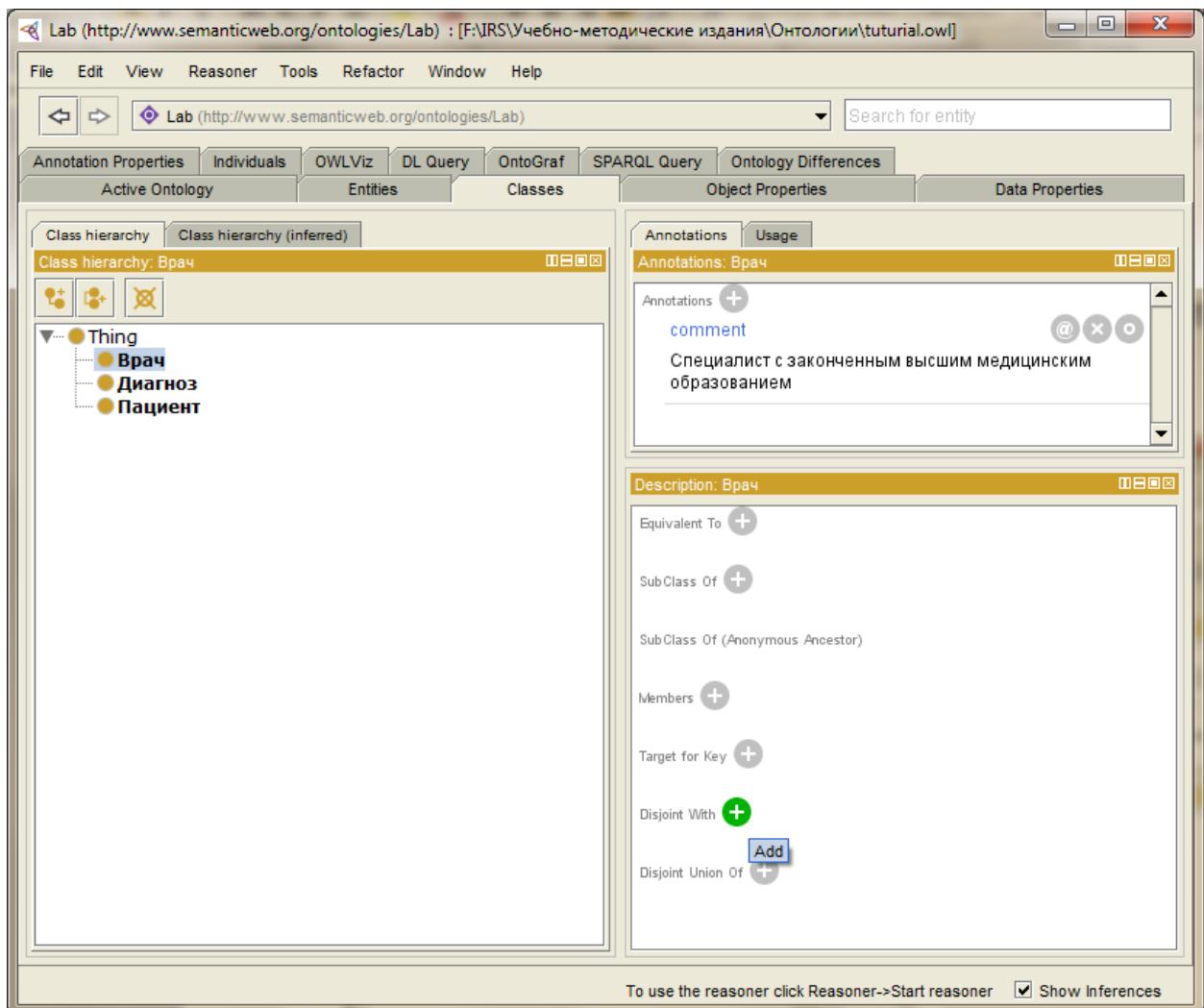


Рисунок 14. Объявление классов непересекающимися

В появившемся окне выберите класс Диагноз (рис. 5.15). Нажмите кнопку «OK».

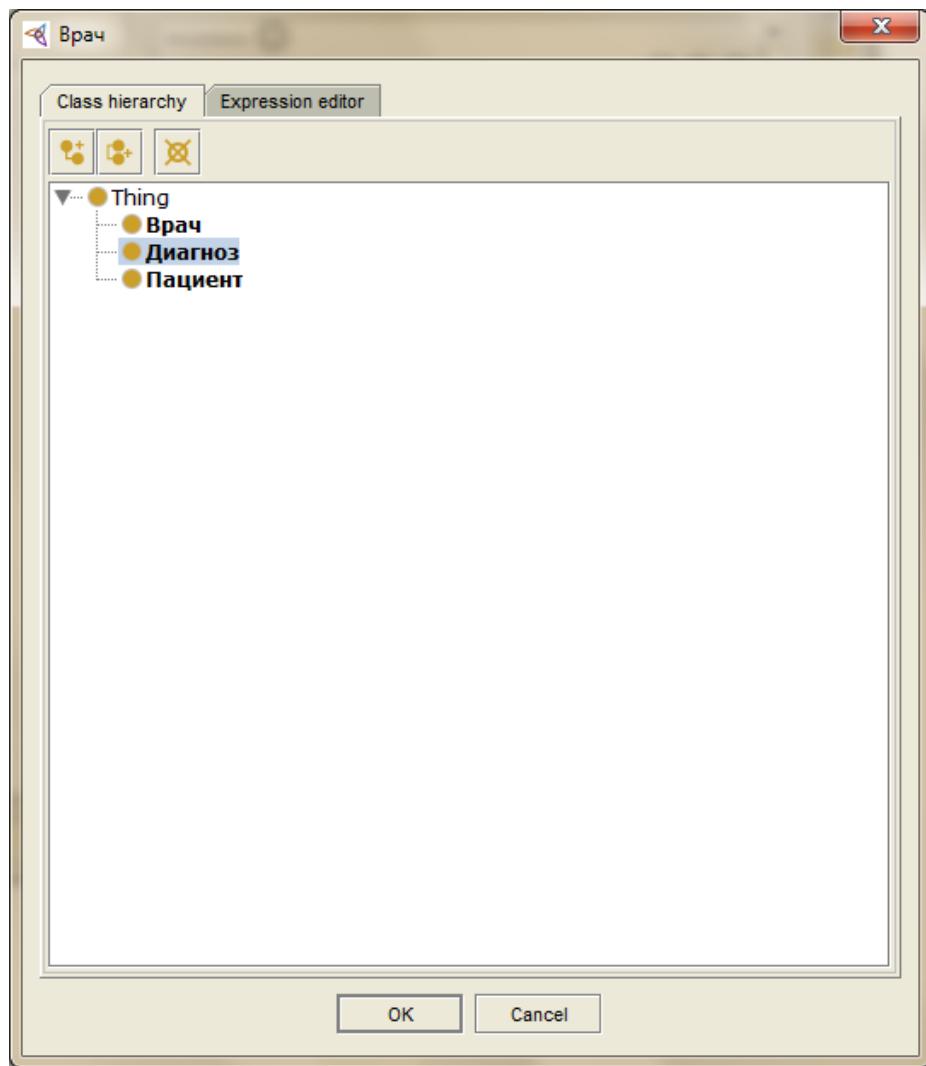


Рисунок 14. Объявление класса Диагноз непересекающимся с классом Врач

В результате выбранные классы онтологии станут непересекающимися. Аналогично сделайте классы **Пациент** и **Диагноз** непересекающимися (рис. 5.15).

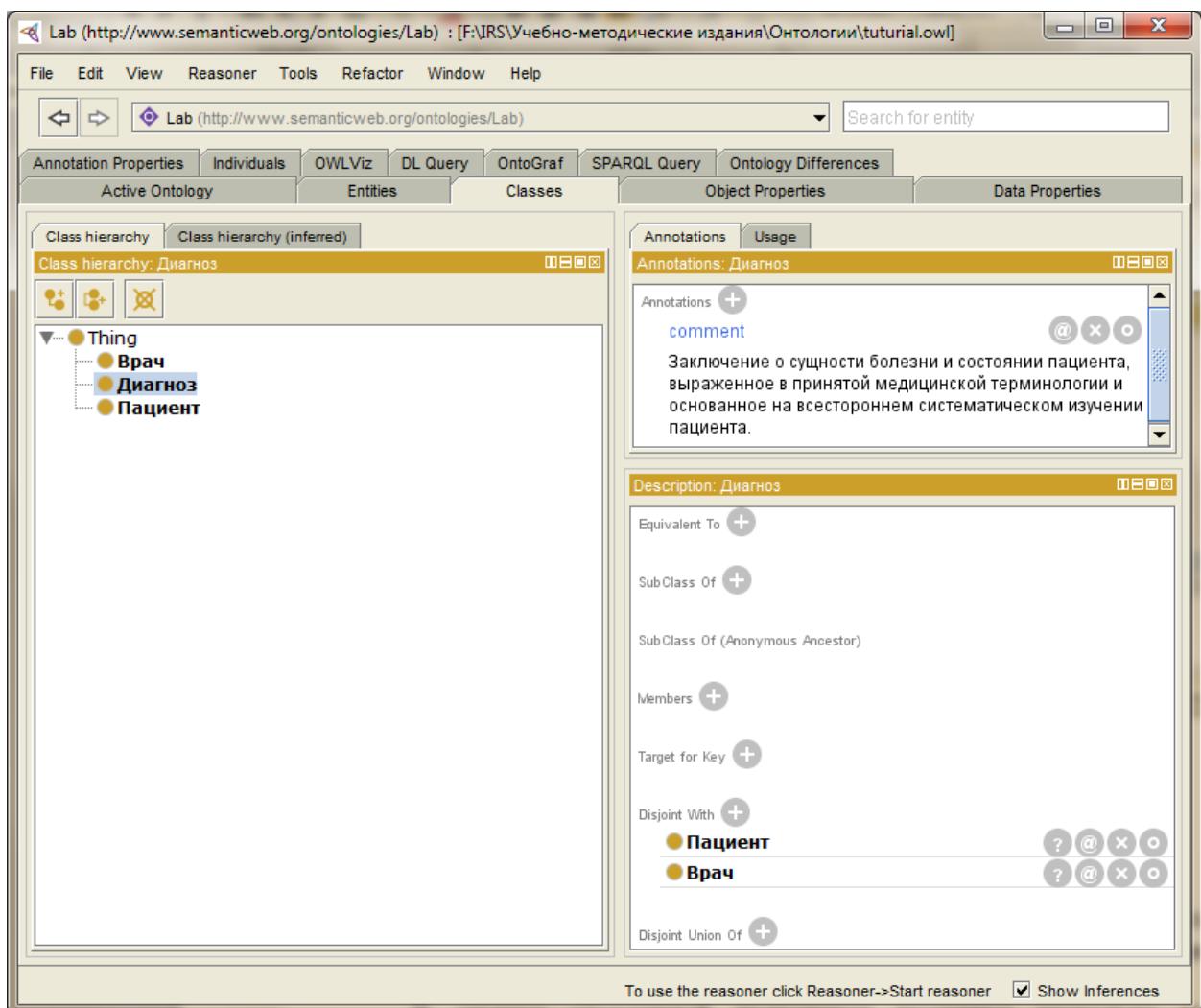


Рис. 5.15 Объявление класса Диагноз непересекающимся с классом Пациент

В диалоговом окне на рис. 5.15 можно выбрать несколько классов, непересекающиеся с данным. Для одновременного выбора нескольких классов нажмите клавишу «Ctrl». В результате выбранные классы онтологии станут попарно непересекающимися.

5.6. Использование мастера создания иерархии классов

Врач осуществляет прием – производимые по определенному плану действия врача при возникновении у пациента потребности в медицинской помощи, представляющие собой сложную или комплексную медицинскую услугу, дающие возможность составить представление о состоянии организма пациента, результатом которых является профилактика, диагностика или лечение определенного заболевания, синдрома.

Создайте класс Прием и напишите к нему аннотацию. Для внесения видов приема в онтологию воспользуемся инструментом «Create Class hierarchy» («Создать иерархию классов»). Для этого выберите пункт меню «Tools» ⇒ «Create Class hierarchy». Появится окно (рис. 5.16), в котором следует указать корневой класс создаваемой иерархии. Выберите класс Прием и нажмите кнопку «Continue».

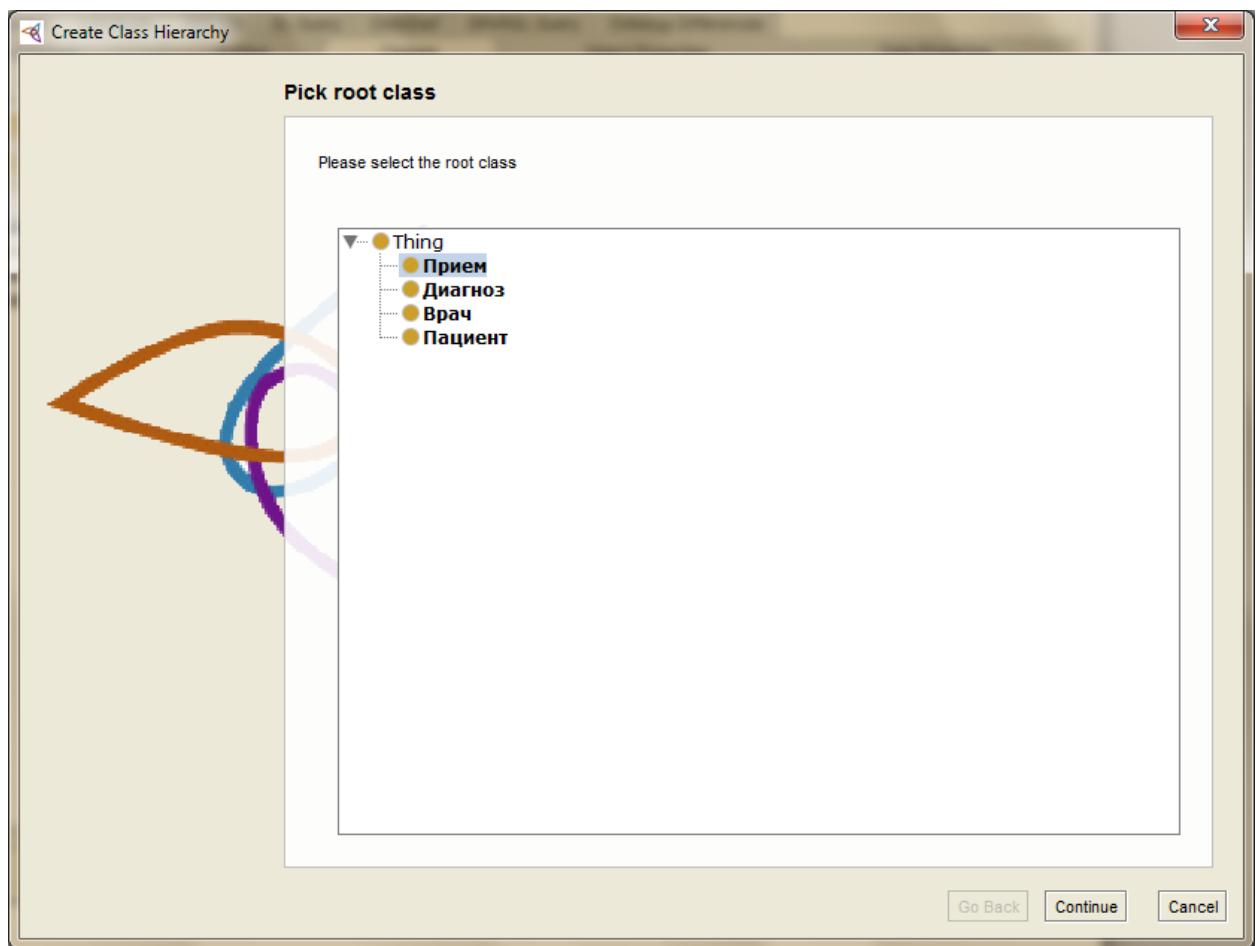


Рис. 5.16. Создание иерархии классов с помощью мастера

Откроется окно создания иерархии классов (рис. 5.17). Поля «Prefix» и «Suffix» используются для автоматического добавления приставки или окончания к именам создаваемых классов. Мастер «Create Class hierarchy» позволяет не только добавить классы к онтологии, но и установить их иераргию с помощью клавиши «Tab». Создайте иерархию классов, как показано на рис. 5.17. Нажмите кнопку «Continue».

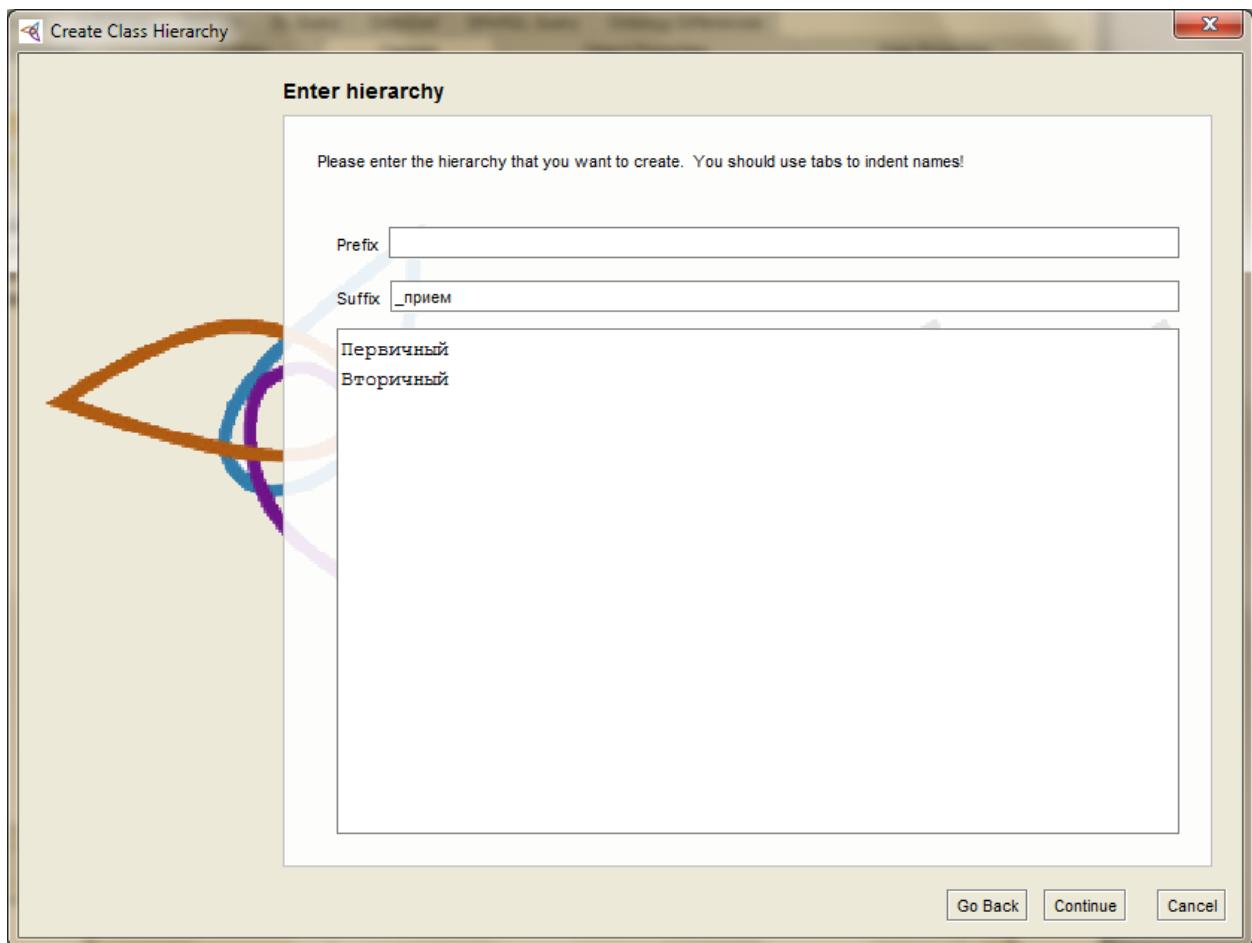


Рис. 5.17 Создание иерархии классов: ввод наименований подклассов

В открывшемся окне (рис. 5.18) убедитесь, что поле «Make sibling classes disjoint» («Сделать одноуровневые классы непересекающимися») отмечено галочкой. Нажмите кнопку «Finish».

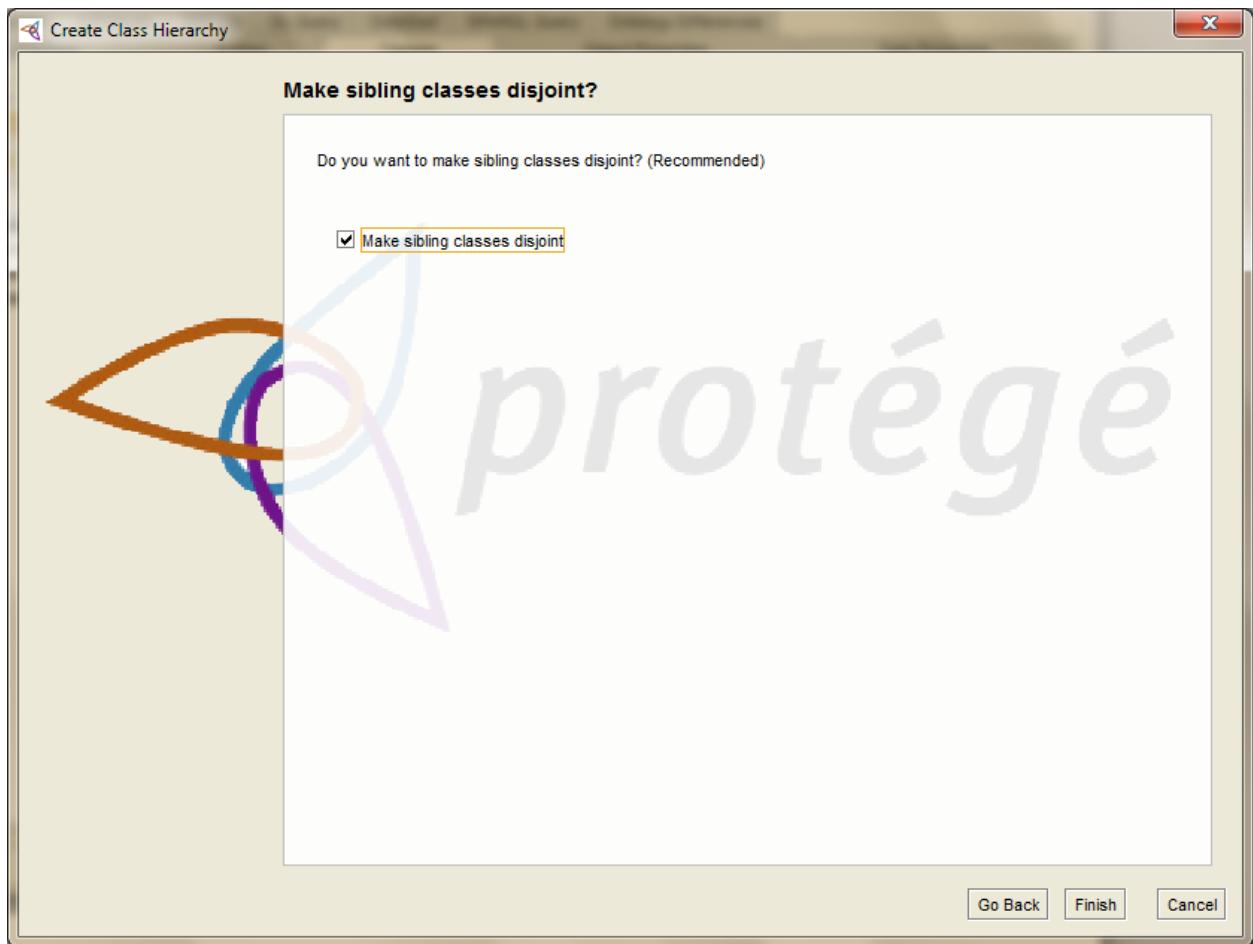


Рис. 5.18 Объявление классов непересекающимися

В результате в онтологию добавлены подклассы класса Прием (рис. 5.19). Обратите внимание, что при выделении подкласса в иерархии классов в поле «Sub Class Of» области «Description» отображается название класса, к которому принадлежит данный подкласс.

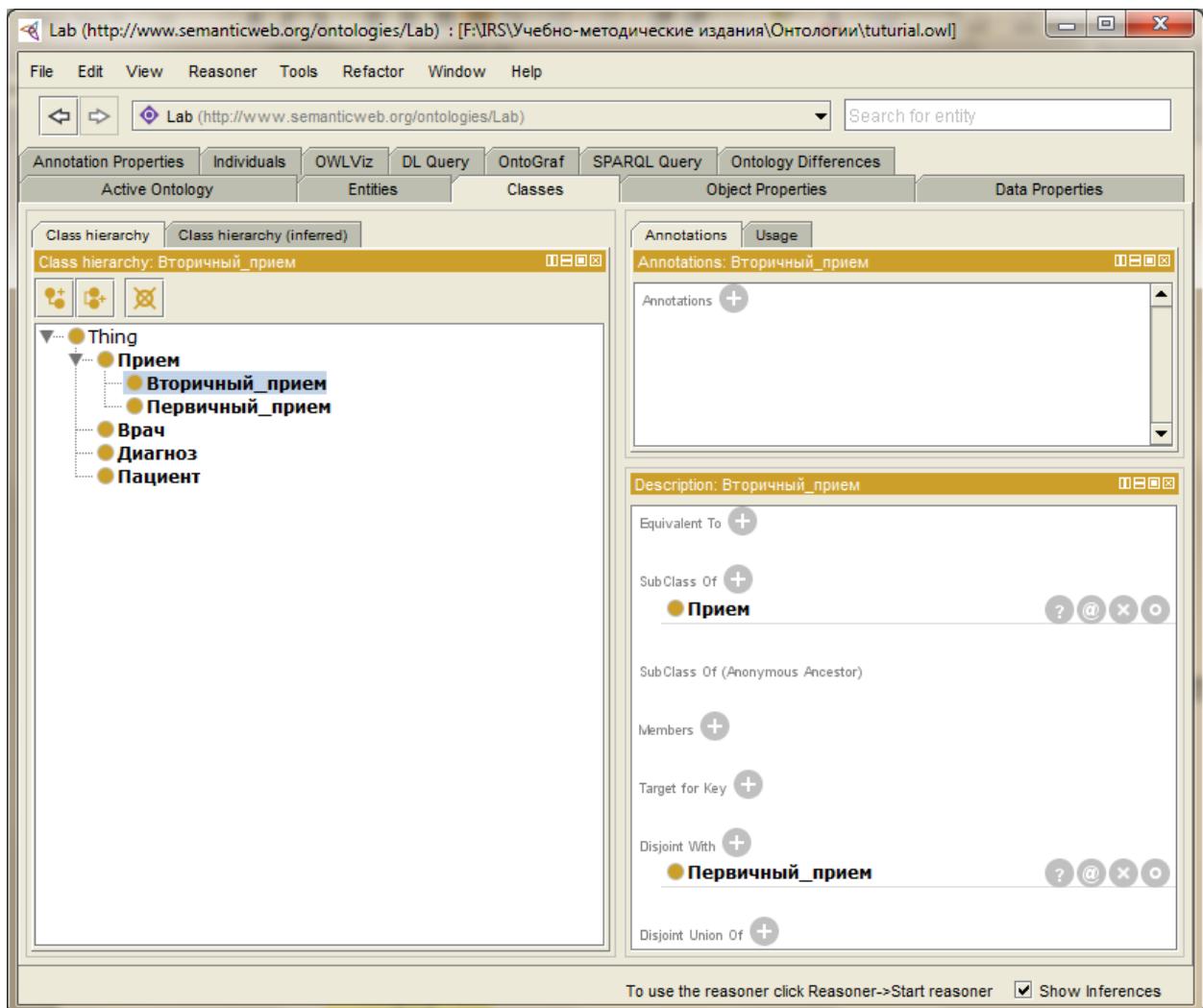


Рис. 5.19. Измененная иерархия классов

Создайте комментарии к подклассам класса **Прием**.

5.7. Свойства онтологии

Свойства онтологии представляют собой бинарные отношения. Существует два основных типа свойств: свойства объектов и свойства данных.

Свойства объектов отражают отношения между двумя классами онтологии. Формально свойство объектов – это бинарное отношение $R_{об} \subseteq K_1 \times K_2$, где K_1, K_2 – классы онтологии. На рис. 5.20 изображен пример свойства объектов.

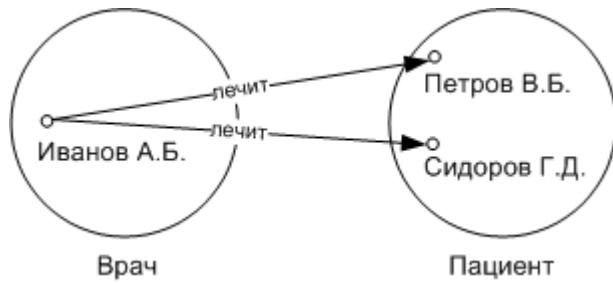


Рис. 5.20. Свойство объектов, связывающее класс Врач и класс Пациент

Свойства данных связывают классы с диапазоном значений, которые может принимать экземпляры классов указанного типа. Формально свойство данных – это бинарное отношение $R_{\text{тд}} \subseteq K \times D$, где K – класс онтологии, D – множество значений определенного типа данных. На рис. 5.21 изображен пример свойства объектов.

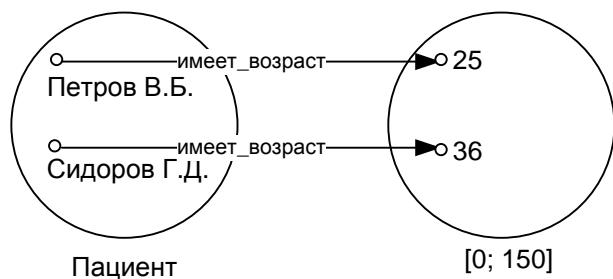


Рис. 5.21. Свойство данных, связывающее класс Пациент и множество [0; 150]

Свойства имеют *домен* (область определения отношения, множество всех первых компонент пар из R) и *диапазон* (область значения отношения, множество всех вторых компонент пар из R). Домены и диапазоны выступают в качестве аксиом при выводе и используются для проверки корректности онтологии.

Свойство может иметь соответствующее обратное свойство, а также обладать свойствами транзитивности, симметричности, асимметричности, рефлексивности, антирефлексивности (иррефлексивности), являться функцией.

Хотя не существует строгого наименования для свойств, рекомендуется имена свойств начинать с маленькой буквы, не допускать пробелов. Также рекомендуется начинать имя свойства с префикса «имеет» или «является», например, `имеет_симптомы`. Также можно формировать имя свойства из наименований связываемых этим свойством множеств. Например, `врач_пациент` – имя отношения, связывающего экземпляры класса Врач с элементами класса Пациент.

5.8. Создание свойства объектов

Переключитесь на вкладку «Свойства объектов» («Object Properties») (рис. 5.22)

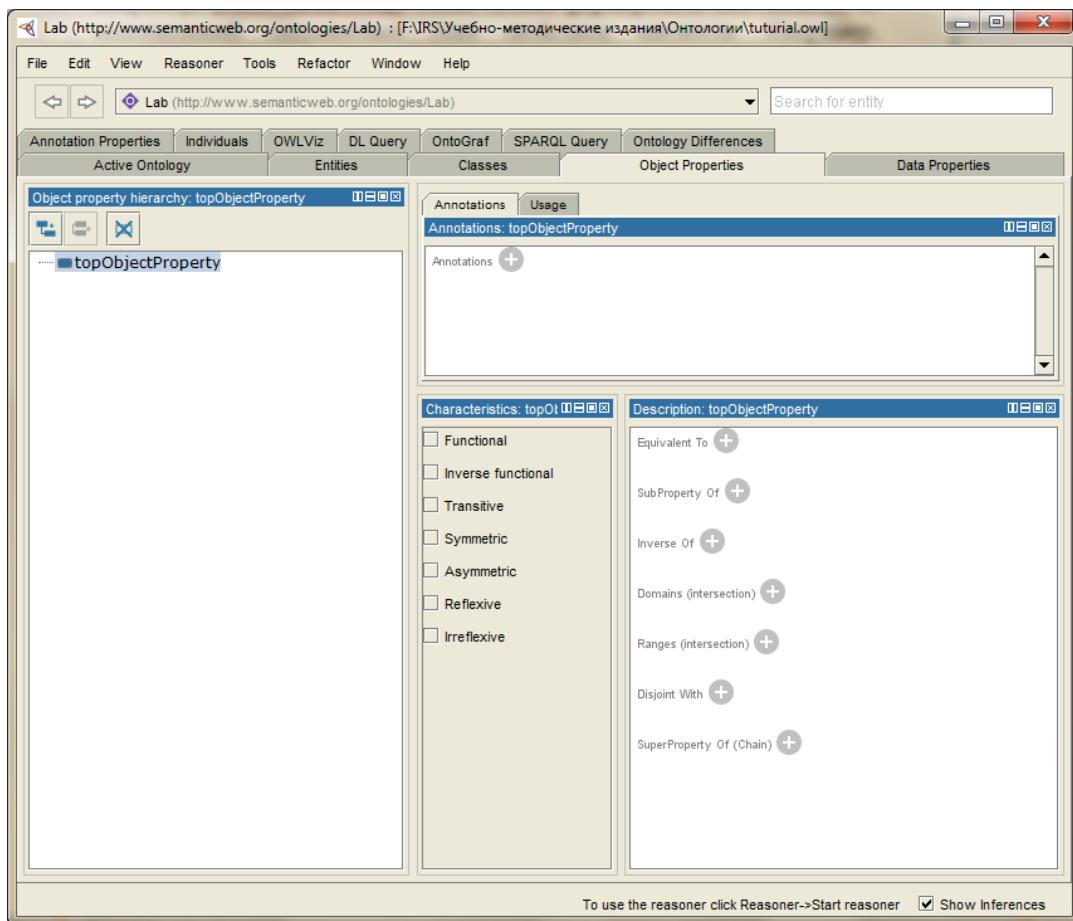


Рис. 5.22. Вкладка «Свойства объектов» («Object Properties»)

Выделите универсальное свойство `topObjectProperty` и нажмите кнопку – «Add sub property» («Создать подсвойство») (рис. 5.23)

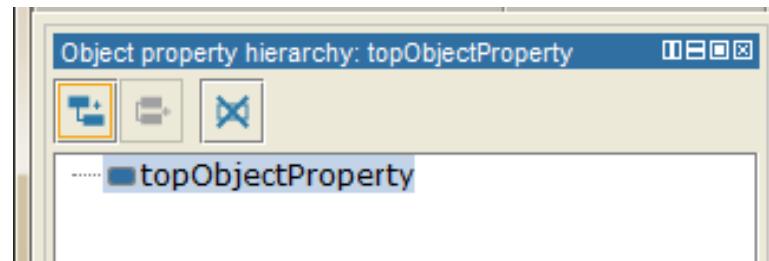


Рис. 5.23 Кнопки для добавления и удаления свойств объектов

Ведите имя свойства в диалоговом окне, как показано на рис. 5.24, нажмите кнопку «OK».

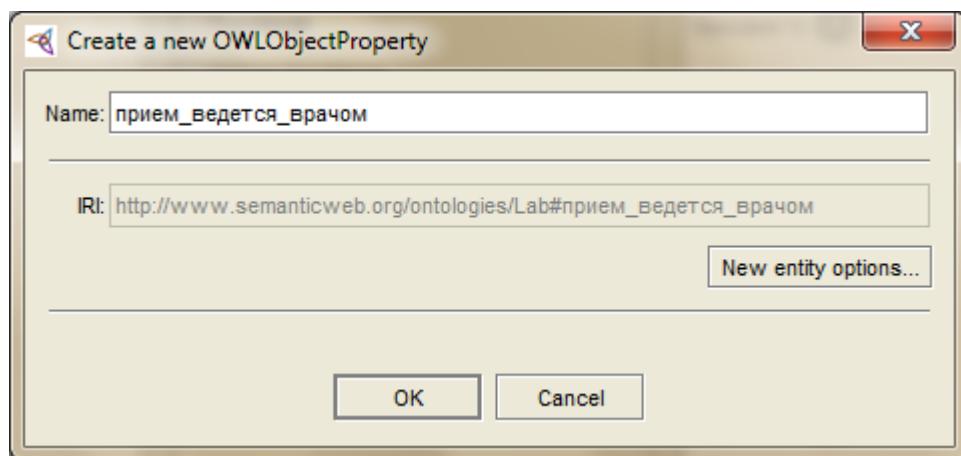


Рис. 5.24. Задание имени свойства

Укажите домен и диапазон отношения (рис. 5.25).

- домен: Прием
- диапазон: Врач

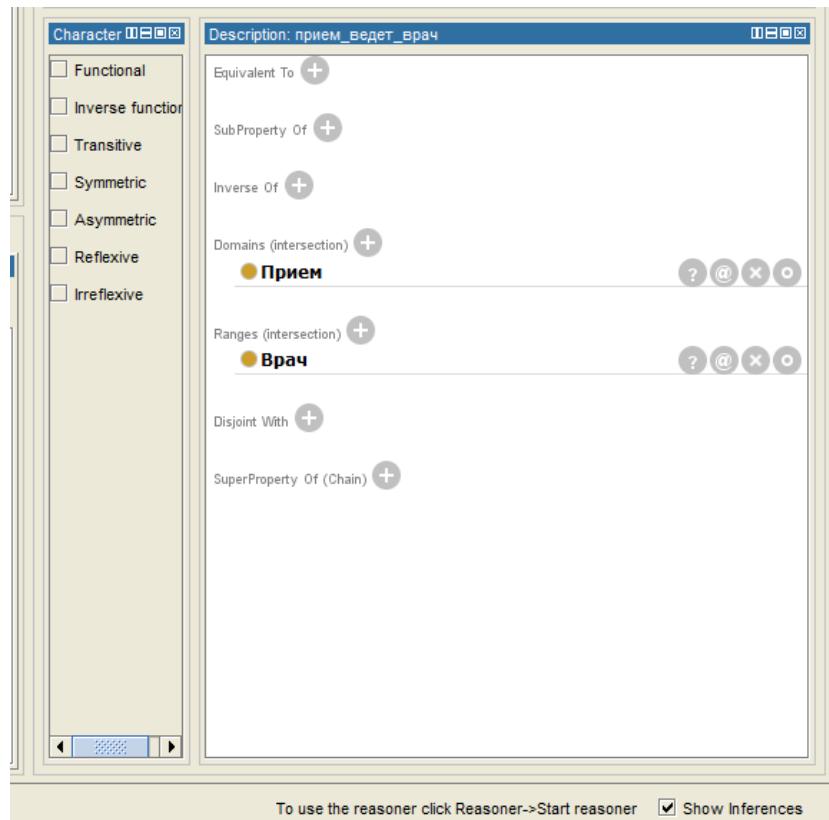


Рис. 5.25. Указание домена и диапазона свойства

Создайте свойство, обратное данному. Для этого на вкладке «Свойства объектов» («Object Properties») создайте новое свойство с именем `врач_ведет_прием`. Затем нажмите значок (+) рядом с заголовком «Inverse Of» (рис. 5.26)

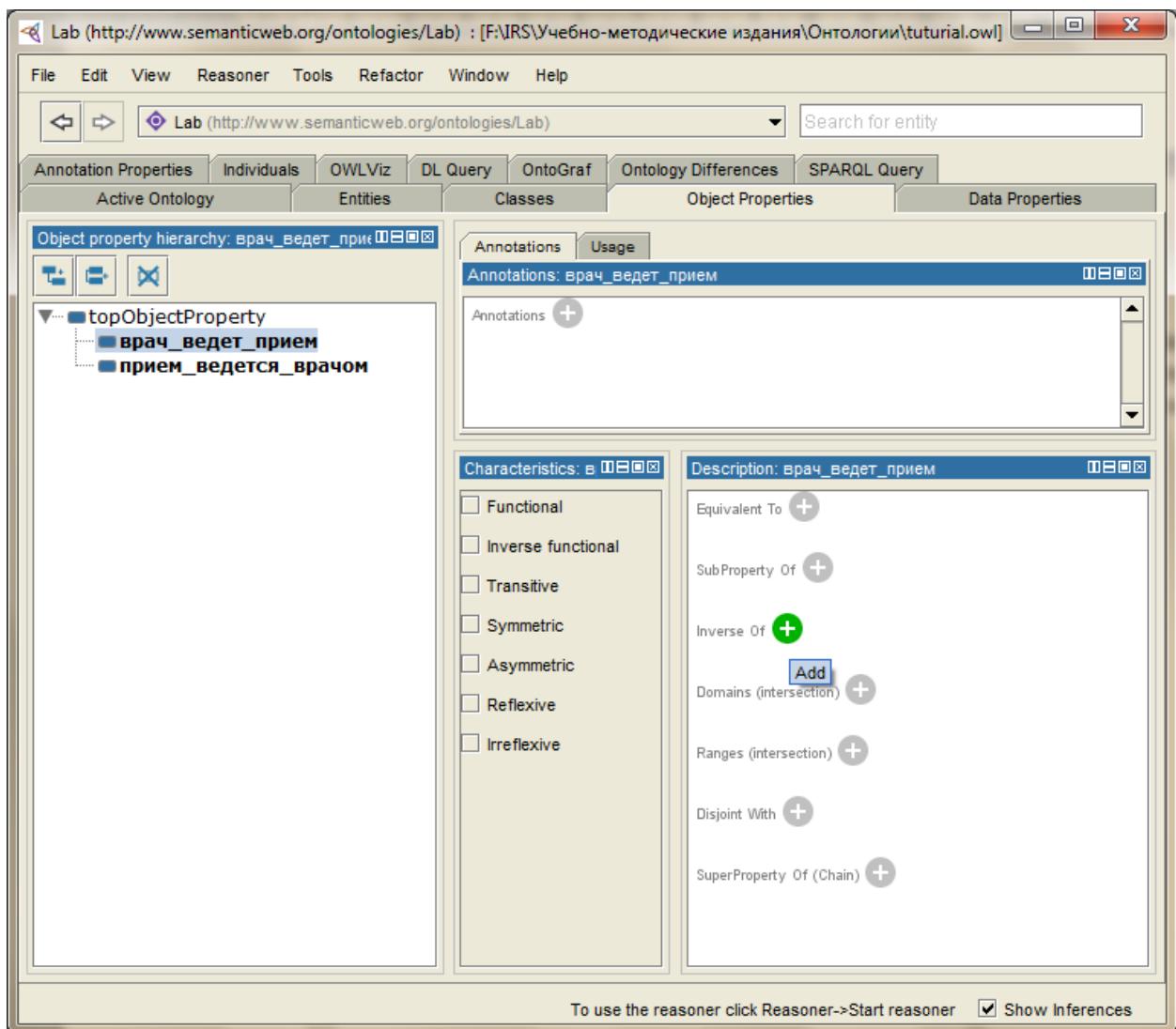


Рис. 5.26. Создание обратного свойства

На экране появится диалоговое окно, в котором в дереве свойств могут быть выбраны нужные свойства. Выберите элемент `прием_ведется_врачом` и нажмите «OK». Свойство `прием_ведется_врачом` теперь должно отображаться как обратное.

В Protégé можно создавать композицию свойств и благодаря этому получать новые отношения. Рассмотрим на примере.

Создайте свойства `пациент_пришел_на_прием` и `пациент_принимается_врачом`. Выберите свойство `пациент_принимается_врачом` в иерархии свойств. Далее на панели «Description» в секции «Super Property Of (Chain)» нажмите кнопку (+). Откроется диалоговое окно текстового редактора, в котором наберите

цепочку «пациент_принимается_врачом о прием_ведется_врачом». В этом выражении знак «о» (английская маленькая буква «о») обозначает операцию композиции двух отношений. В этом редакторе также можно использовать технику автозаполнения.

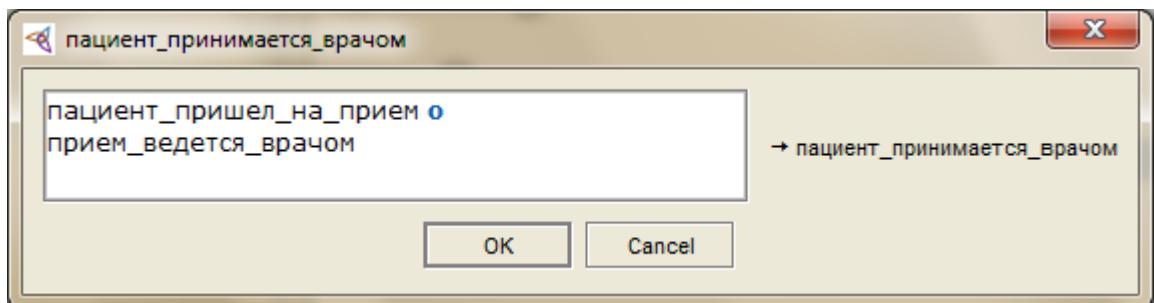


Рис. 5.27. Создание композиции свойств

Композиция отношений позволяет в более естественной и краткой форме описывать свойства объектов с помощью логических формул.

5.9. Виды ограничений

В Protégé можно формулировать утверждения о классах и их экземплярах. Семантика онтологии задается путем интерпретации ограничений – высказываний о ее классах и свойствах. Ограничение может быть задано в форме необходимых условий (в секции Sub Class Of) и в форме необходимых и достаточных условий (в секции Equivalent To). В случае необходимых условий описание говорит, что если экземпляр принадлежит классу, то он обладает указанными свойствами. Класс, который удовлетворяет только необходимым ограничениям, известен как *примитивный класс*. В случае необходимых и достаточных условий описание, кроме вышеуказанного смысла, несет еще информацию, что если экземпляр обладает заданными свойствами, то он принадлежит описываемому классу.

Ограничения разделяют на три основные категории:

- кванторные ограничения
- ограничения мощности
- ограничения на свойства данных

Квантор – это общее название для логических операций, ограничивающих область истинности какого-либо. В Protege используются квантор всеобщности \forall и квантор существования \exists .

Ограничение, полученное навешиванием квантора всеобщности, называют *универсальным ограничением*. В Protégé для его обозначения используют ключевое слово *only*. Ограничение, полученное навешиванием квантора существования, называют *экзистенциальным ограничением*. В Protégé для его обозначения используют ключевое слово *some*.

Ограничения класса можно просматривать и редактировать, используя панель «Описание класса» («Description») вкладки «Классы».

5.10. Создание кванторных ограничений

Экзистенциальные ограничения являются самым распространенным типом ограничений в OWL. Создадим ограничение, которое определяет подкласс врачей, проводящих операции. Для этого создайте класс Операция и свойство врач_проводит_операцию.

Выберите Врач из иерархии классов на вкладке «Классы». Нажмите «Добавить» – значок (+) рядом с заголовком «Sub Class Of» в панели «Описание класса». Это позволит открыть диалоговое окно редактора ограничений, в котором можно создать ограничение с помощью имеющихся классов и свойств. В открывшемся окне перейдите на вкладку «Object restriction creator». Для создания ограничения нужно сделать следующее (рис. 5.28):

- Выбрать указанное свойство из списка свойств;
- Ввести тип ограничения (*some*) для экзистенциального ограничения;
- Указать требуемый класс как аргумент ограничения.

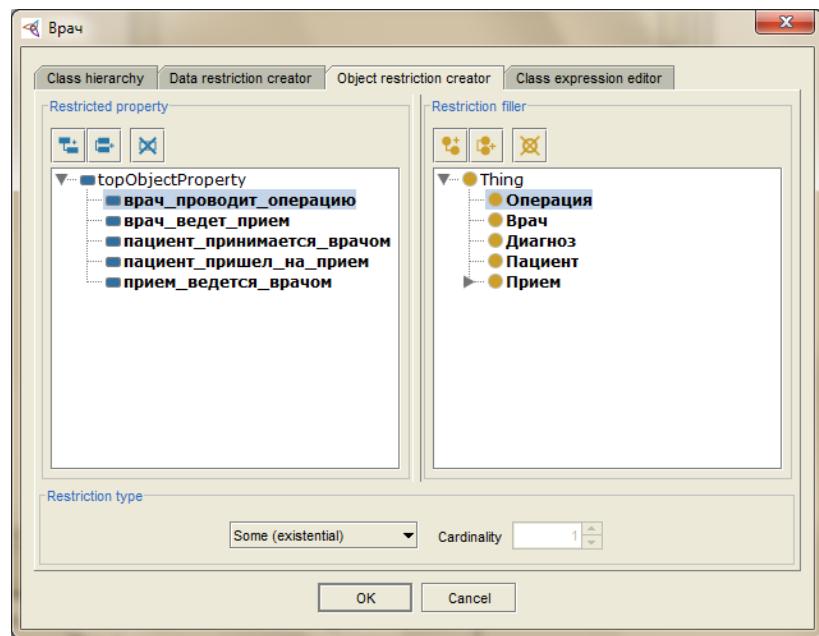


Рис. 5.28. Окно создания ограничений

Результат представлен на рис. 5.29.

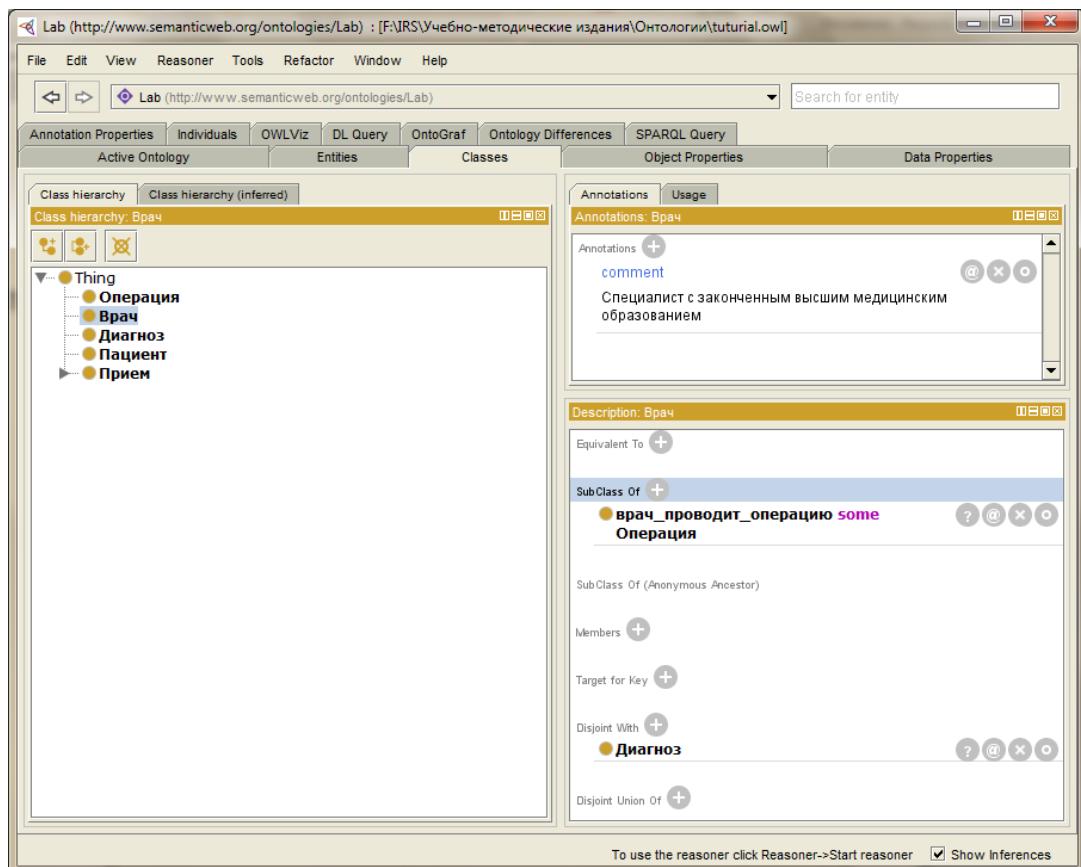


Рис. 5.29. Создание экзистенциального ограничения

Задать ограничение можно также на вкладке «Class expression editor» (рис. 5.30).

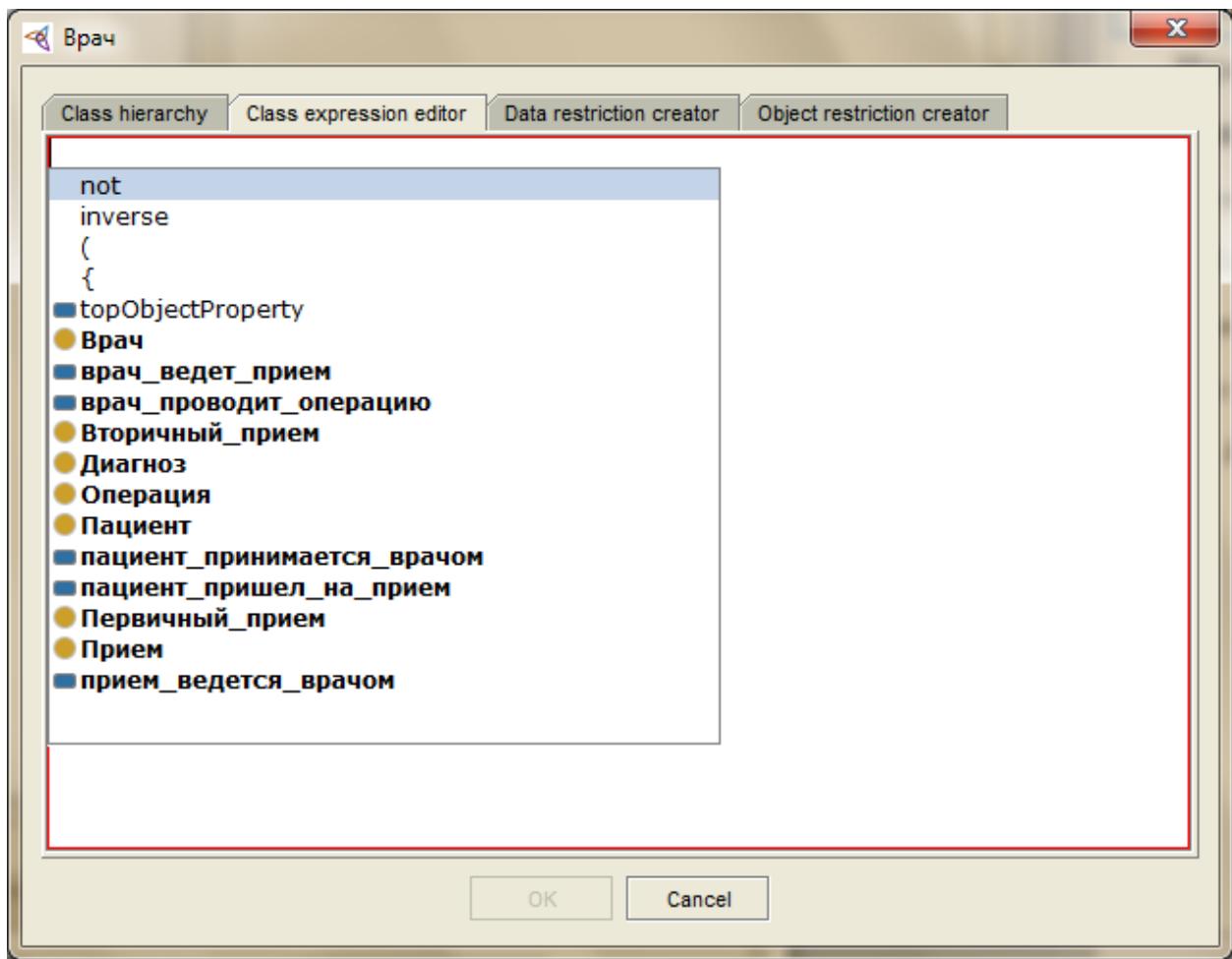


Рис. 5.30. Создание ограничения с помощью построителя выражений

Текстовое окно позволяет создать ограничение с помощью имен классов, свойств и экземпляров. При нажатии на сочетание клавиш «Ctrl-Пробел» активируется построитель выражений. Он предоставляет возможность «автозаполнения» имен классов, свойств и экземпляров

Интерпретация описанного выше ограничения, помещенного в область Sub Class Of, порождает множество, принадлежность которому описывается *необходимым* условием: «Если экземпляр является членом этого класса, то она соответствует указанным ограничениям». При этом мы не можем сказать, что «если сущность удовлетворяет этим условиям, то она должна быть членом этого класса».

Класс, который удовлетворят только необходимым условиям, известен как *примитивный* класс. Класс, который удовлетворяет необходимым и достаточным условиям, называется *определяемым*. Чтобы

построить определяемый класс, следует ограничение поместить в область Equivalent To.

Рассмотрим класс Врач. Что быть допущенным к профессиональной деятельности врача, медицинскому работнику необходимо и достаточно иметь врачебную специальность. Создадим данное ограничение.

- Создайте класс Врачебная_специальность.
- Создайте свойство врач_имеет_специальность
- Нажмите «Добавить» – значок (+) рядом с заголовком «Equivalent To» и задайте соответствующее экзистенциальное ограничение (рис. 5.31)

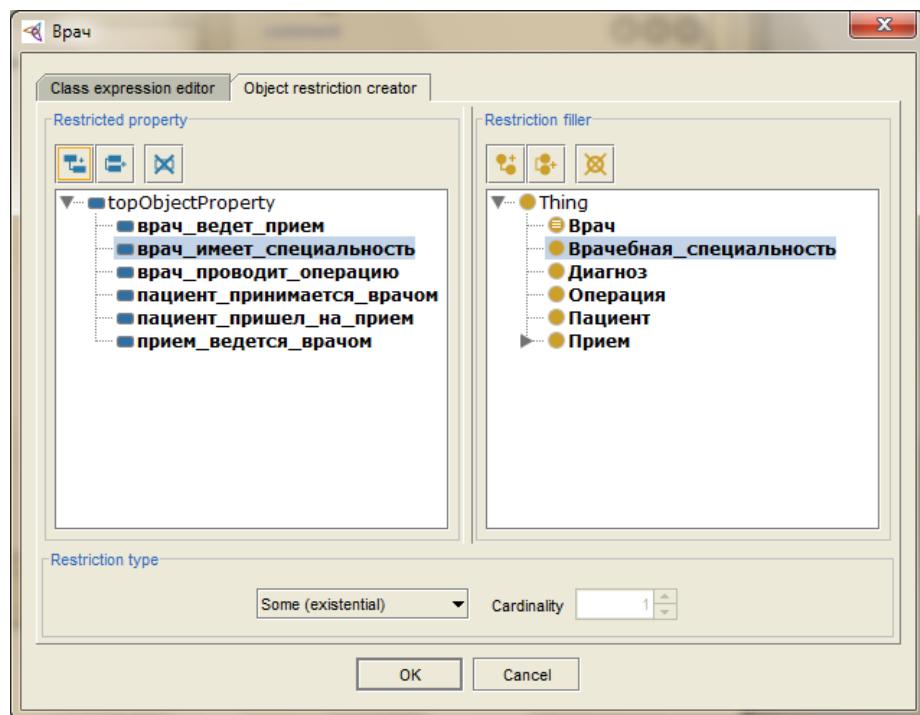


Рис. 5.31. Создание ограничения

Результат представлен на рис. 5.32:

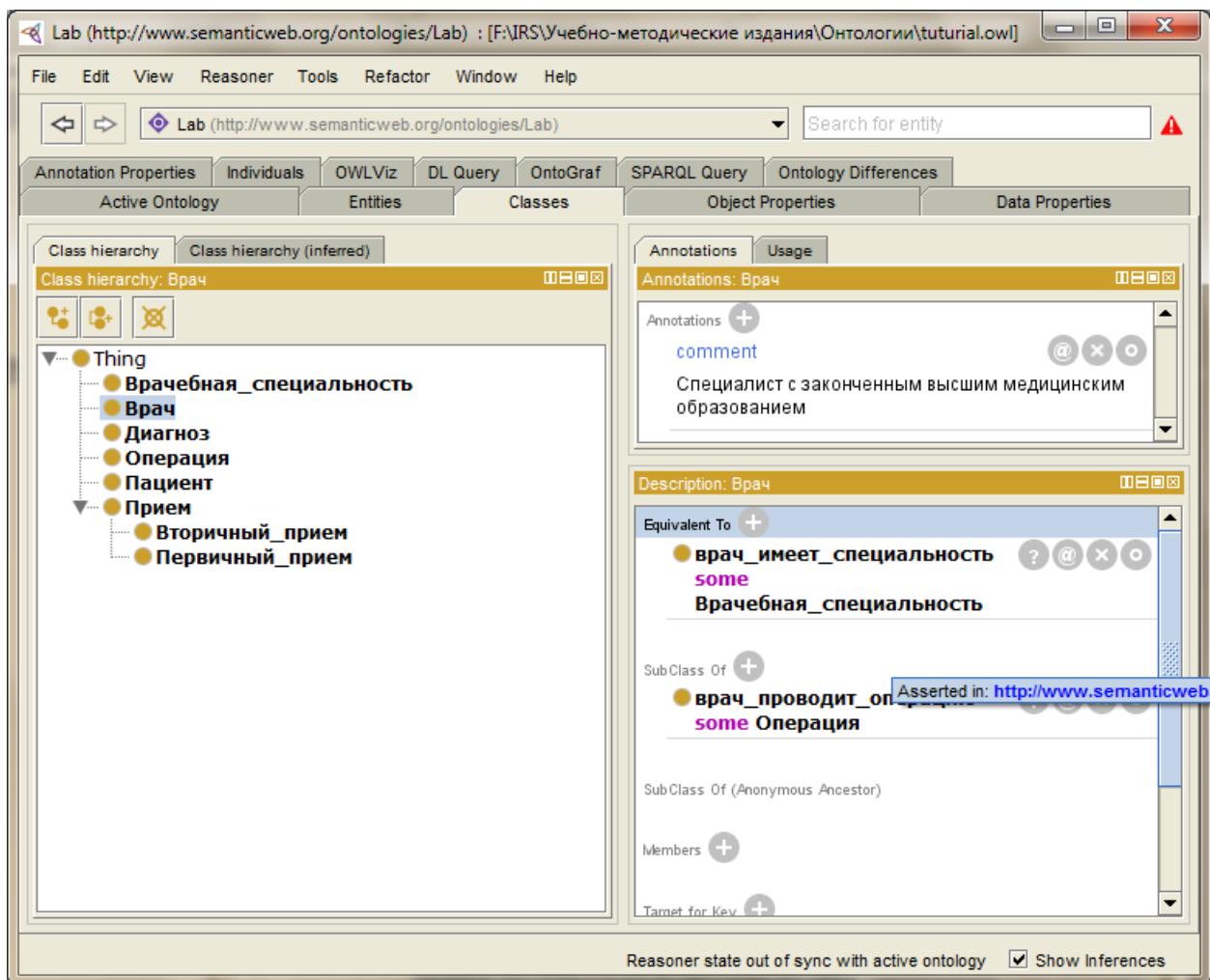


Рис. 5.31. Задание определяемого класса

Универсальные ограничения создаются аналогично. Им соответствует ключевое слово `only`.

5.11. Использование машины вывода для проверки непротиворечивости иерархии классов

Одной из ключевых особенностей онтологий, описываемых с помощью OWL, является возможность обработки машиной вывода (резонером). Одно из предназначений машины вывода в том, чтобы проверить, действительно ли один класс является подклассом другого класса. Еще одна полезная функция резонера – проверка согласованности ограничений с помощью автоматических рассуждений. При этом выполняется вывод иерархии классов онтологии. Класс считается противоречивым, если его ограничениям не удовлетворяет ни

одна сущность. Таким образом, машина вывода классифицирует объекты предметной области и проверяет согласованность онтологии.

Protégé позволяет подключать различные резонеры к редактору. Иерархия классов, которая построена вручную, называется *присоединенной иерархией*. Иерархия классов, которая автоматически вычисляется в процессе рассуждений, называется *выводимой иерархией*.

Чтобы автоматически классифицировать онтологию и проверить ее на непротиворечивость, следует выбрать в главном меню Reasoner⇒Start reasoner. Если класс будет признан несовместимым, то он будет выделен ярким цветом.

Для того, чтобы продемонстрировать работу машины вывода в обнаружении несоответствий в онтологии, создадим тестовый класс, который определим как подкласс двух непересекающихся классов.

- Выберите класс Врач.
- Создайте подкласс класса Врач с именем Тест.
- Выберите в иерархии классов класс Тест. В секции Sub Class Of нажмите на кнопку «Добавить» (+). В появившемся диалоговом окне выберите класс «Диагноз» и нажмите «OK» (рис. 5.32)

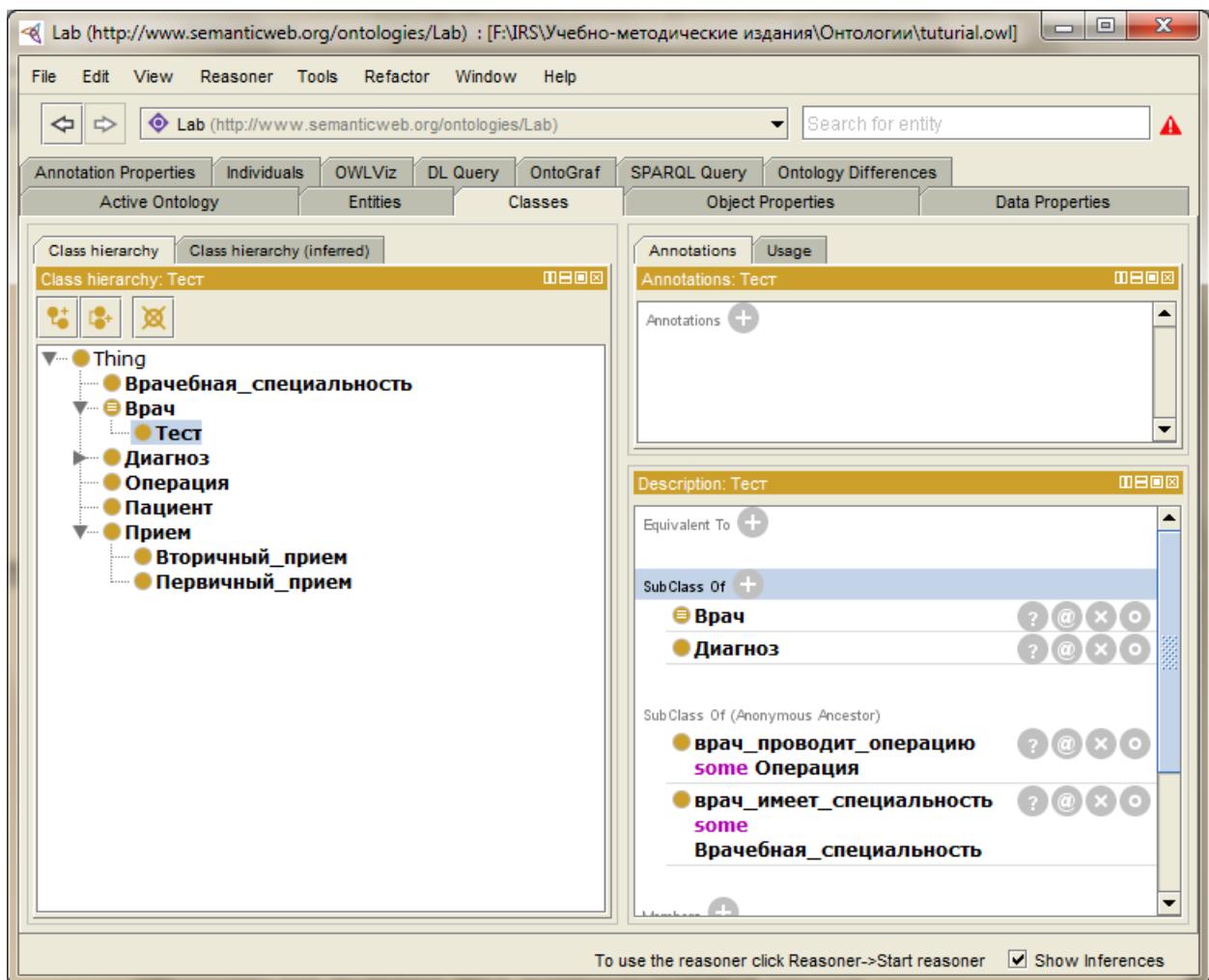


Рис. 5.32. Автоматическая классификация

Таким образом, класс Тест является подклассом двух непересекающихся классом. Это возможно лишь в том случае, если Тест – пустой класс.

Выберите Reasoner⇒Start reasoned. Выводимую иерархию можно увидеть на вкладке «Class hierarchy» (рис. 5.33).

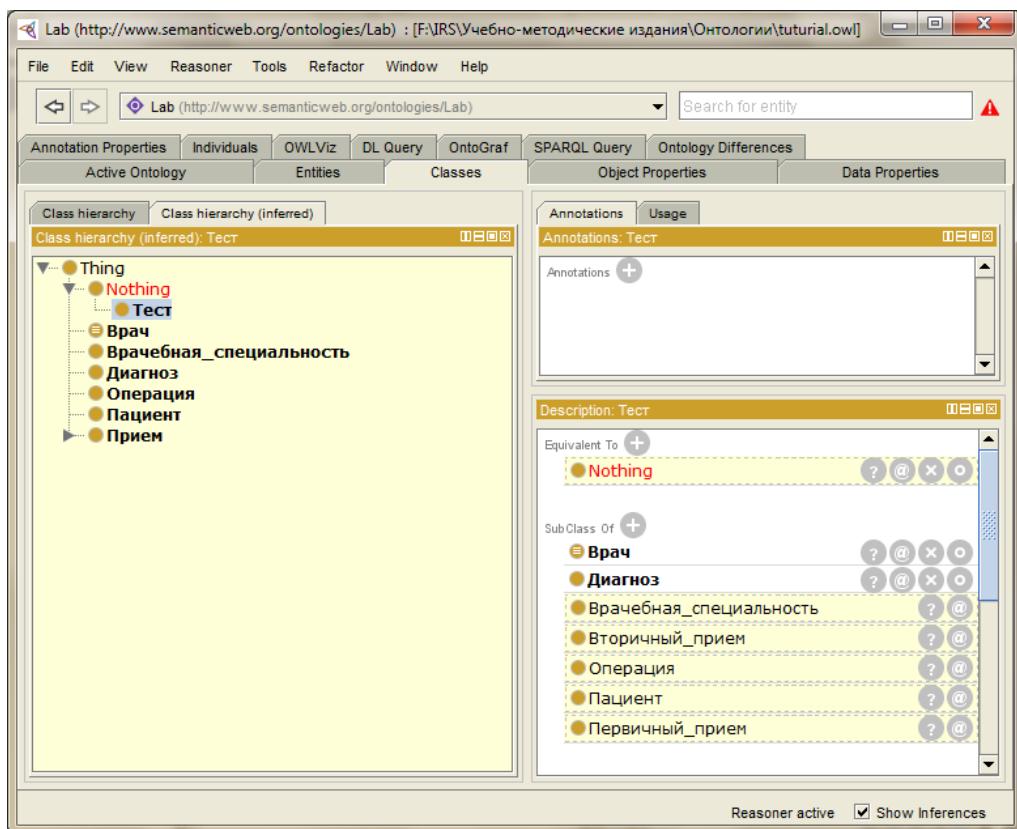


Рис. 5.33. Выводимая иерархия

Как видно, класс Тест признан резонером несоместимым с онтологией (т.е. он не может иметь экземпляров). Вернитесь к первоначальному варианту онтологии. выполните классификацию резонером и убедитесь в непротиворечивости онтологии.

5.12. Описание ограничений мощности

Мощность ограничения используется для описания количества экземпляров заданного отношения, в которых объект может присутствовать. Существуют три варианта мощности ограничений: минимальная мощность ограничения (min), максимальная мощность ограничения (max), точное значение мощности ограничения (exactly).

Для каждого пациента заводится медицинская карта, причем в единственном экземпляре. Опишем данное ограничение в Protégé.

- Создайте класс Медицинская_карта.
- Создайте свойство пациент_имеет_медкарту.

Выделите класс Пациент и нажмите значок (+) в разделе «Equivalent to». В диалоговом окне создайте ограничение пациент_имеет_медкарту exactly 1 Медицинская_карта (рис. 5.34).

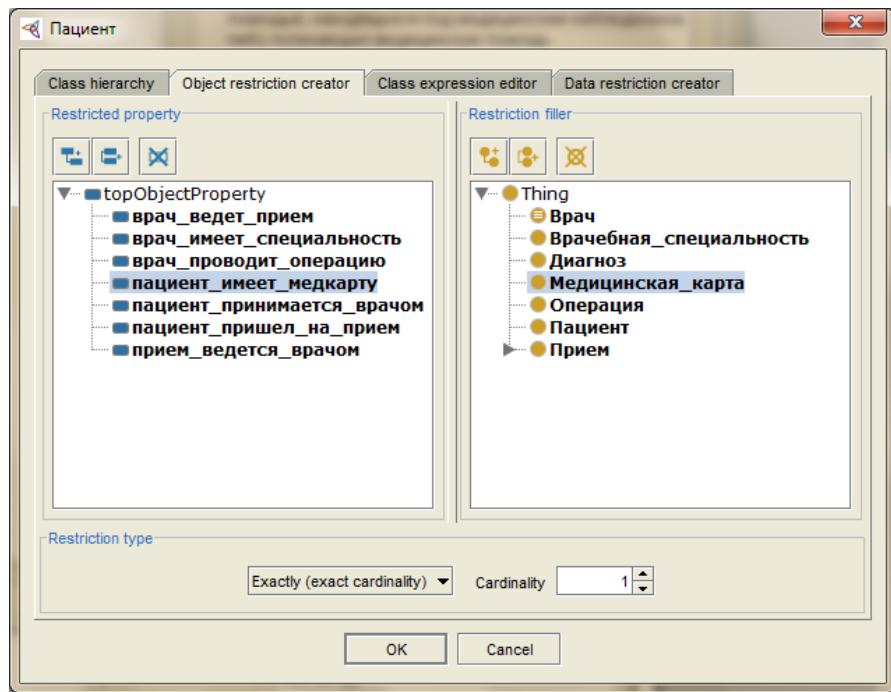


Рис. 5.34. Создание ограничения мощности

Описание класса Пациент представлено на рис. 5.35.

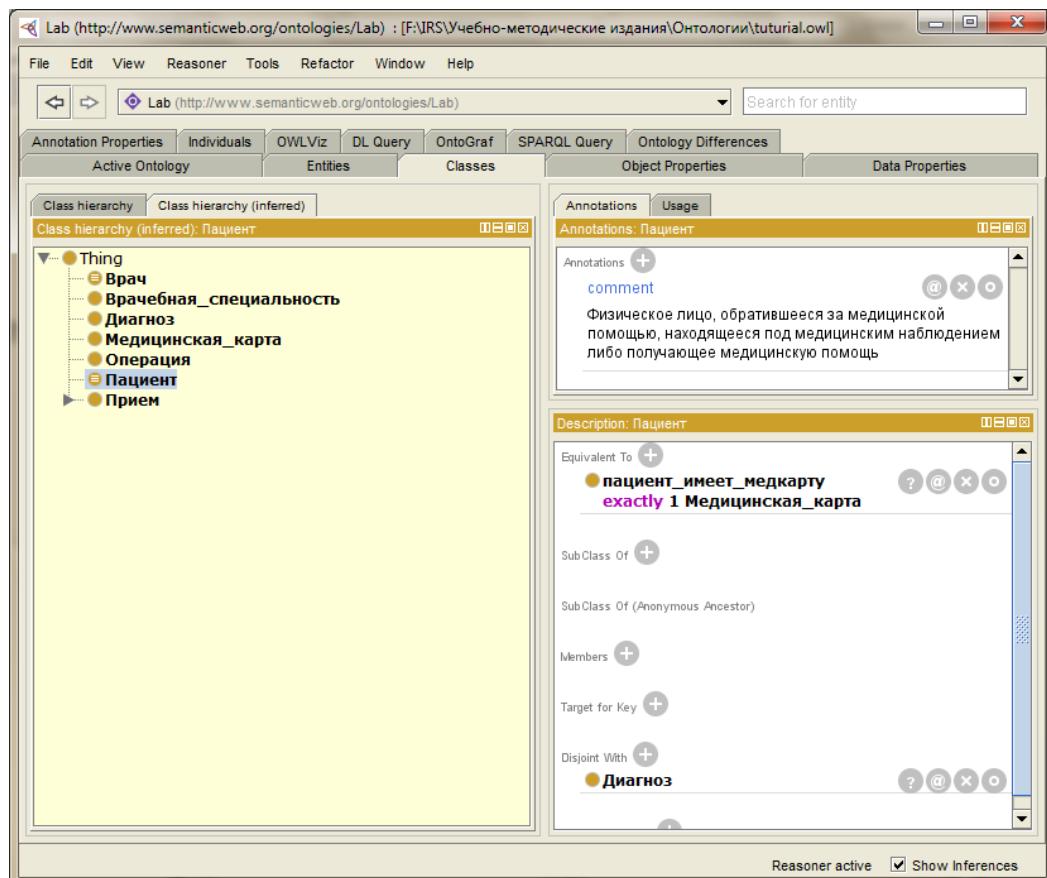


Рис. 5.35. Задание класса Пациент ограничением мощности

5.13. Свойства типа данных

Иерархия классов не позволяет составить полное представление о предметной области. Поэтому после определения классов необходимо описать отличительные свойства, присущие их экземплярам – свойства типа данных, которые описывают связи между характеристиками экземпляров классов и значениями типов данных.

Свойства типа данных могут быть созданы с помощью вкладки **Data Properties**, показанной ниже (рис. 5.36).

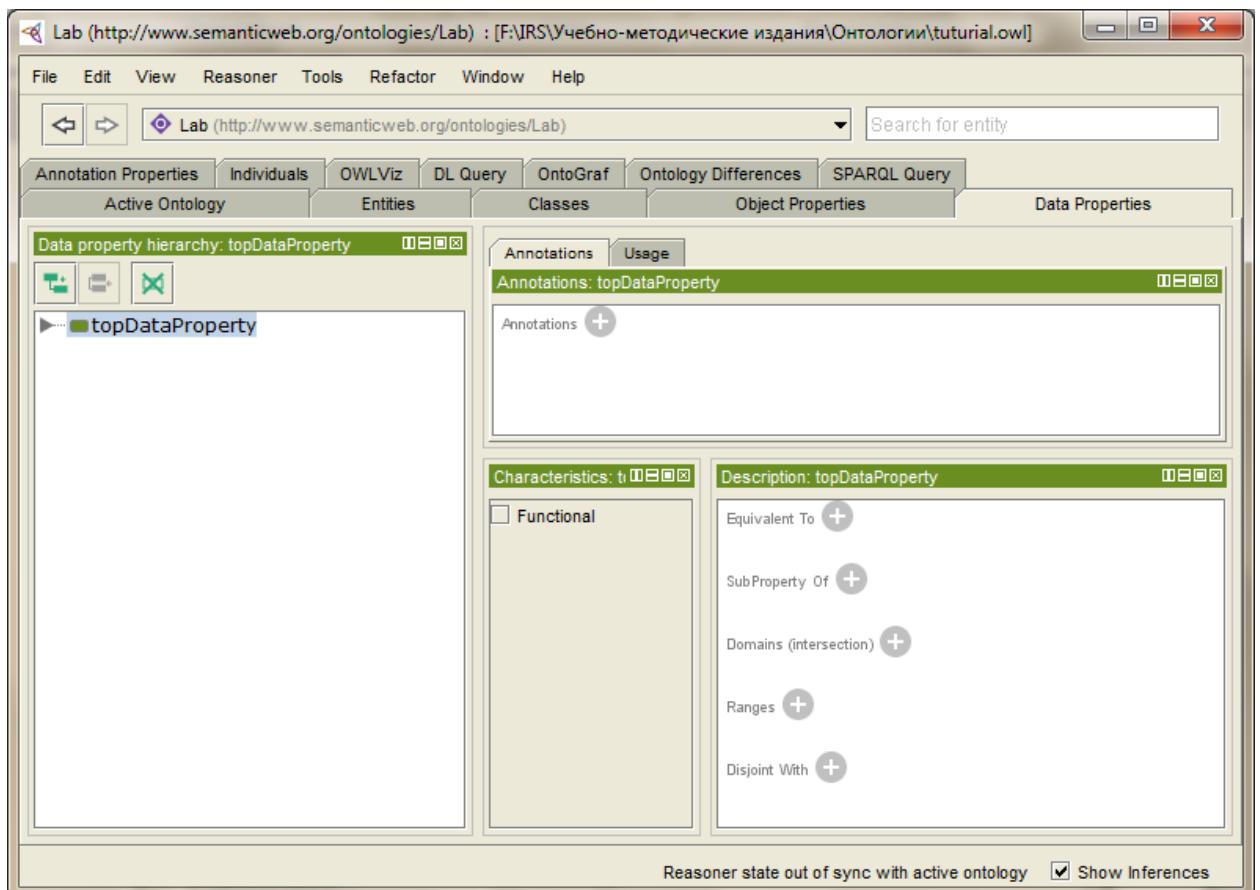


Рис. 5.35. Вкладка «Свойства типа данных» («Data Properties»)

Будем использовать свойства данных для описания возраста пациентов:

- Переключитесь на вкладку «Data Properties».
- Создайте новое свойство данных имеет_возраст (рис. 5.36)

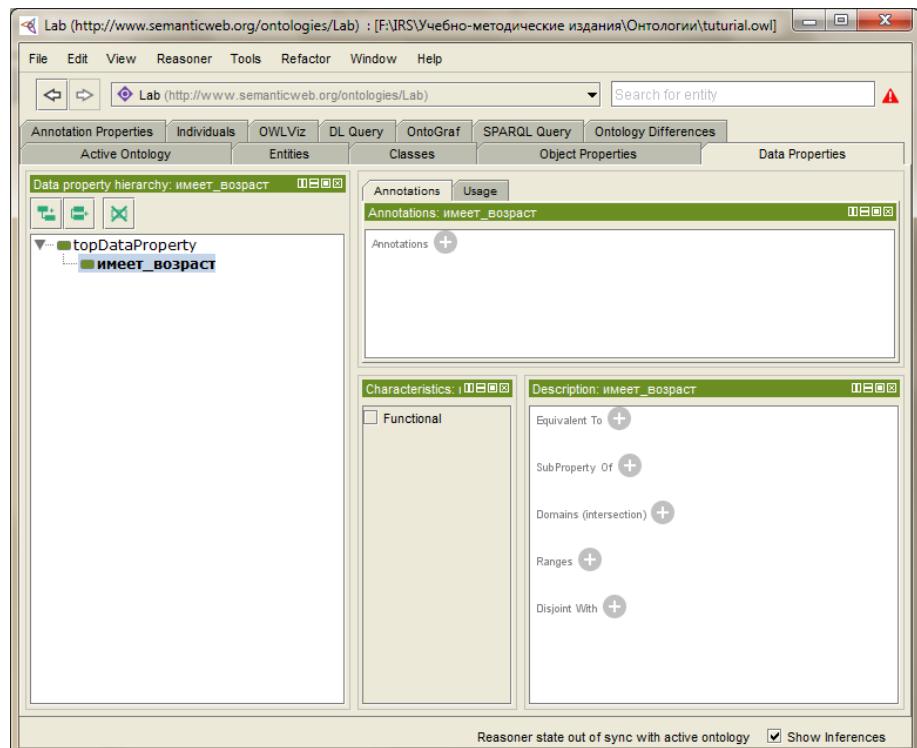


Рис. 5.35. Создание свойства типа данных имеет_возраст

Создадим экзистенциальное ограничение, устанавливающее, что все экземпляры класса Пациент обладают свойством имеет_возраст (рис. 5.36):

- На вкладке «Классы» выберите класс Пациент.
- Нажмите кнопку «Добавить» (+) в секции «Sub Class Of».
- В открывшемся диалоговом окне выберите вкладку «Data restriction creator» («Создать ограничение на свойство данных»).
- В правой части выберите свойство имеет_возраст.
- Установите тип ограничения some.
- В правой части укажите тип данных integer.

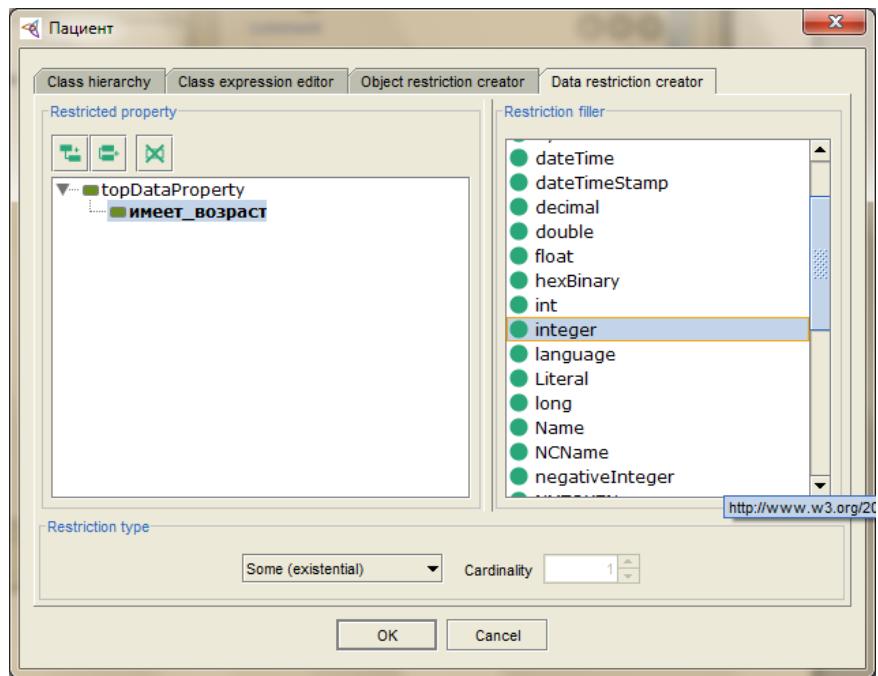


Рис. 5.35. Создание ограничения

Нажмите «OK». Созданное ограничение появилось в секции «Sub Class Of» (рис. 5.36).

Рис. 5.35. Описание класса Пациент экзистенциальным ограничением

После создания свойства типа данных необходимо описать экземпляров класса, обладающих этим свойством. Экземпляры класса имеют изначально при создании только имя, а потом при описании у них появляется набор свойств, который может быть абсолютно индивидуальным. Задание экземпляров классов и присваивание им набора свойств происходит на вкладке «Individuals»:

- На вкладке «Individuals» нажмите кнопку  – «Добавить экземпляр» и введите его имя – Петров_В.Д.
- В панели «Description» («Описание») укажите тип – Пациент.
- В панели «Property assertions» («Определение связей») выберите секцию «Data property assertions» и нажмите кнопку «Добавить». В появившемся диалоговом окне укажите в качестве свойства данных имеет_возраст, в качестве типа данных – integer и в текстовом поле введите значение возраста пациента 42 (рис. 5.36).

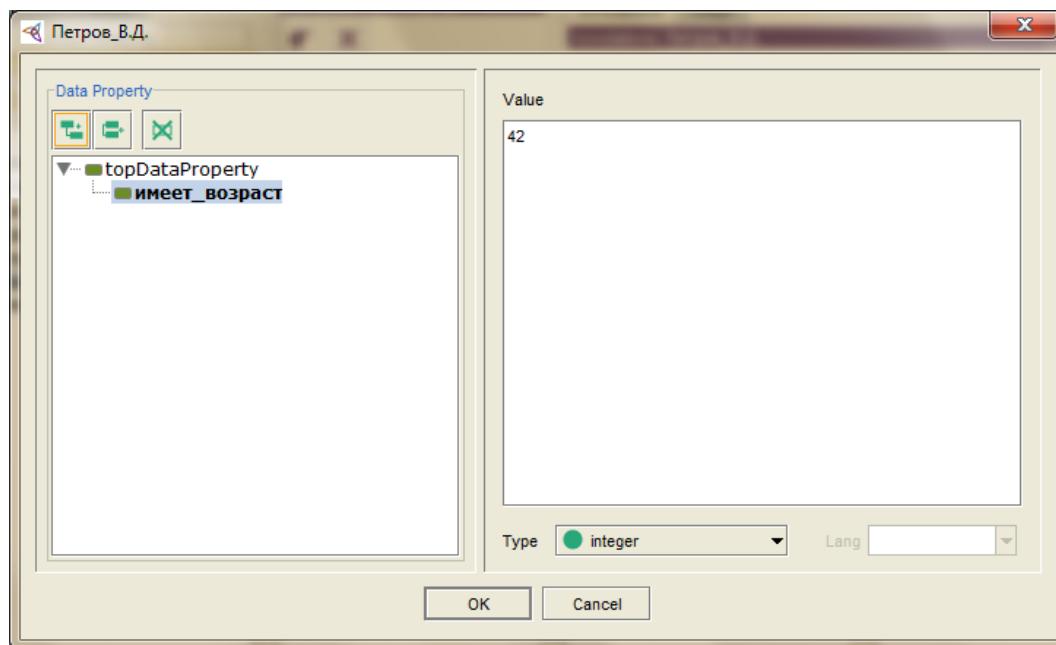


Рис. 5.36. Задание экземпляра класса

Создайте еще несколько экземпляров класса Пациент: Иванов М.А. – 20 лет, Сидоров К.Л. – 65 лет.

5.14. Описание ограничений на свойства типа данных

Введем несколько возрастных интервалов для классификации пациентов по возрастным группам. Используя типизированные данные, можно задавать ограничения вида \leq , \geq , $<$, $>$, $=$ на их значения. Например:

Молодой_пациент \equiv имеет_возраст some integer [< 25]

Пожилой_пациент \equiv имеет_возраст some integer [≥ 60]

Пациент_средних_лет \equiv имеет_возраст some integer [≥ 25 , < 60].

Создадим первое ограничение. Для этого на вкладке «Classes» создайте подкласс Молодой_пациент класса Пациент. На панели «Description» в секции «Sub Class Of» нажмите кнопку «Добавить» (+). В редакторе выражений наберите имеет_возраст some integer [< 25] и нажмите «OK». Выделите созданное ограничение и выберите Edit \rightarrow Convert to defined class для конвертации примитивного класса в определяемый. Описание класса Молодой_пациент будет иметь вид, представленный на рис. 5.37.

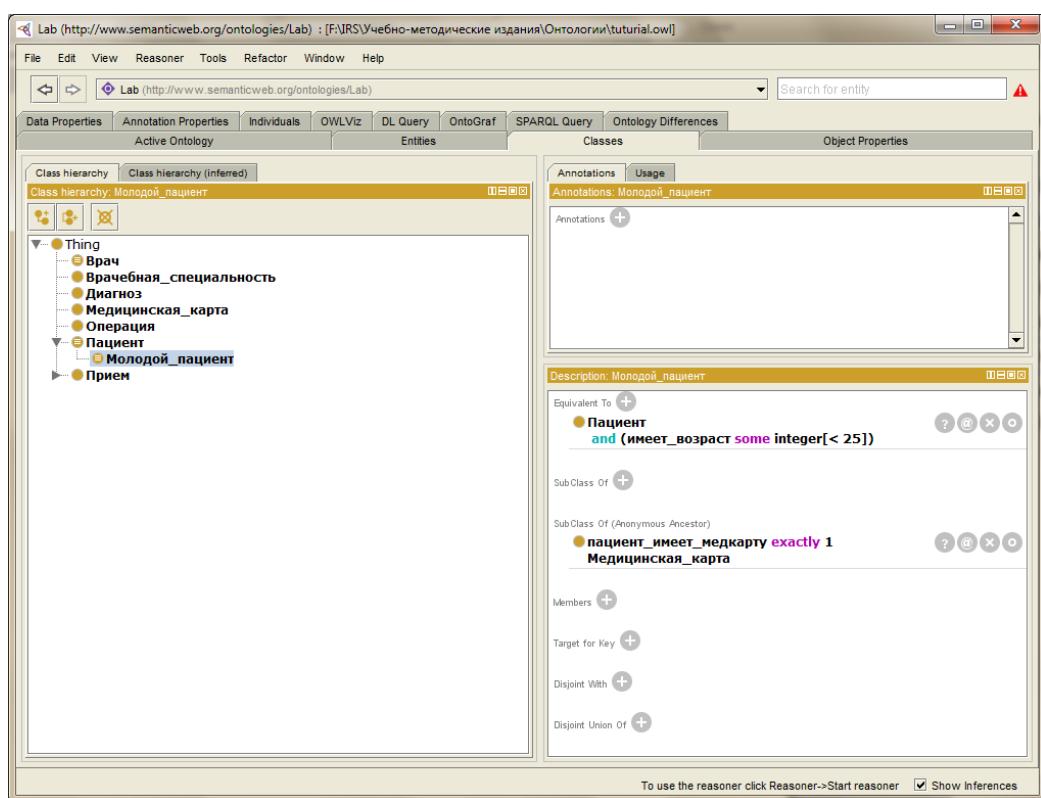


Рис. 5.37. Создание ограничения на свойство типа данных

Аналогично создайте подкласс Пожилой_пациент и Пациент_средних_лет.

Классифицируйте пациентов по возрасту с помощью резонера (рис. 5.38):

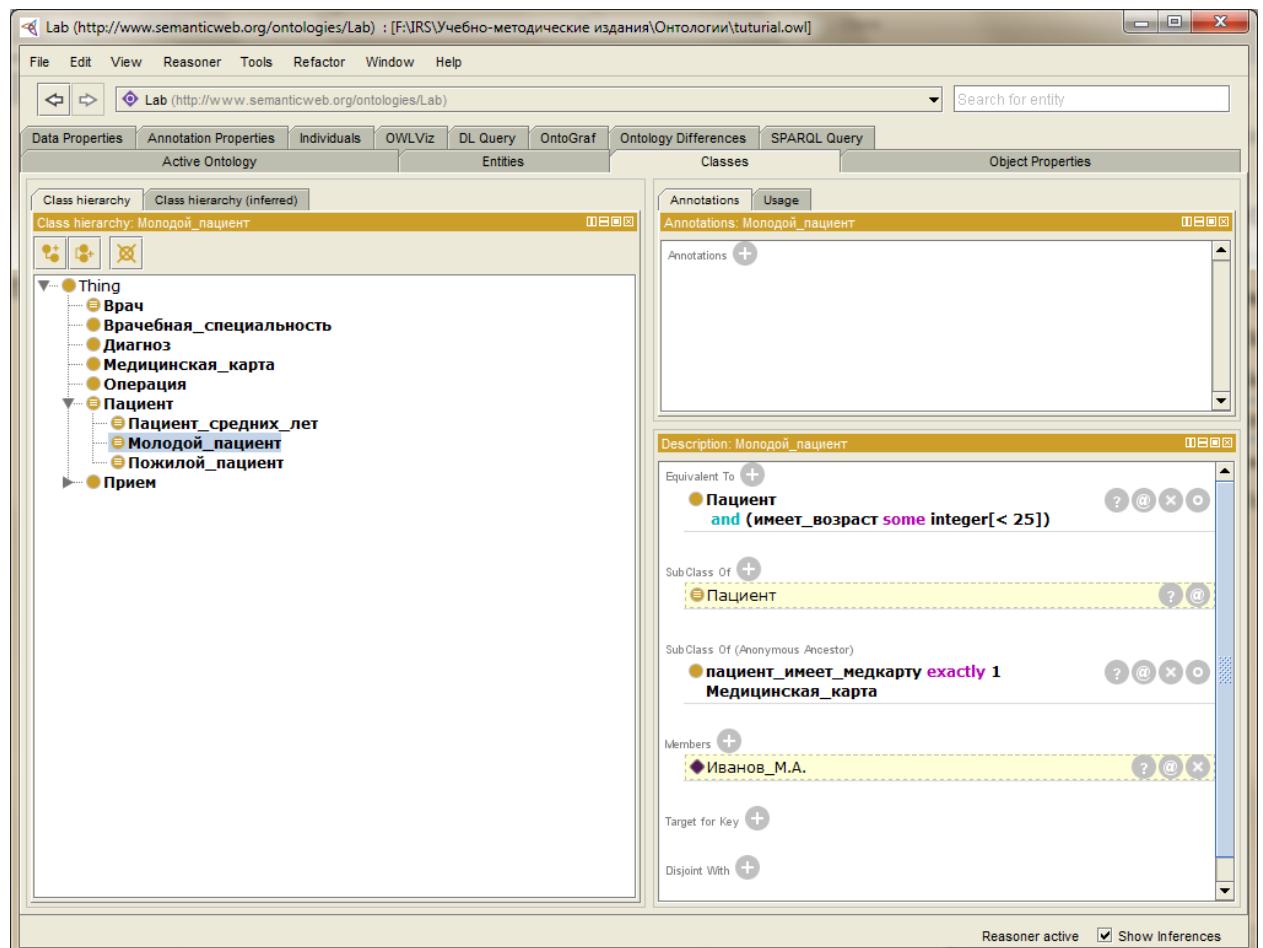


Рис. 5.38. Классификация пациентов

Обратите внимание на секцию «Members». Она должна включать экземпляры, имеющие возраст в границах указанного диапазона

5.15. Создание DL-QUERY запросов

Для создания запросов необходимо перейти в закладку DL-Query, а также на вкладке «Reasoner» выбрать «Start reasoner» (рис. 5.39).

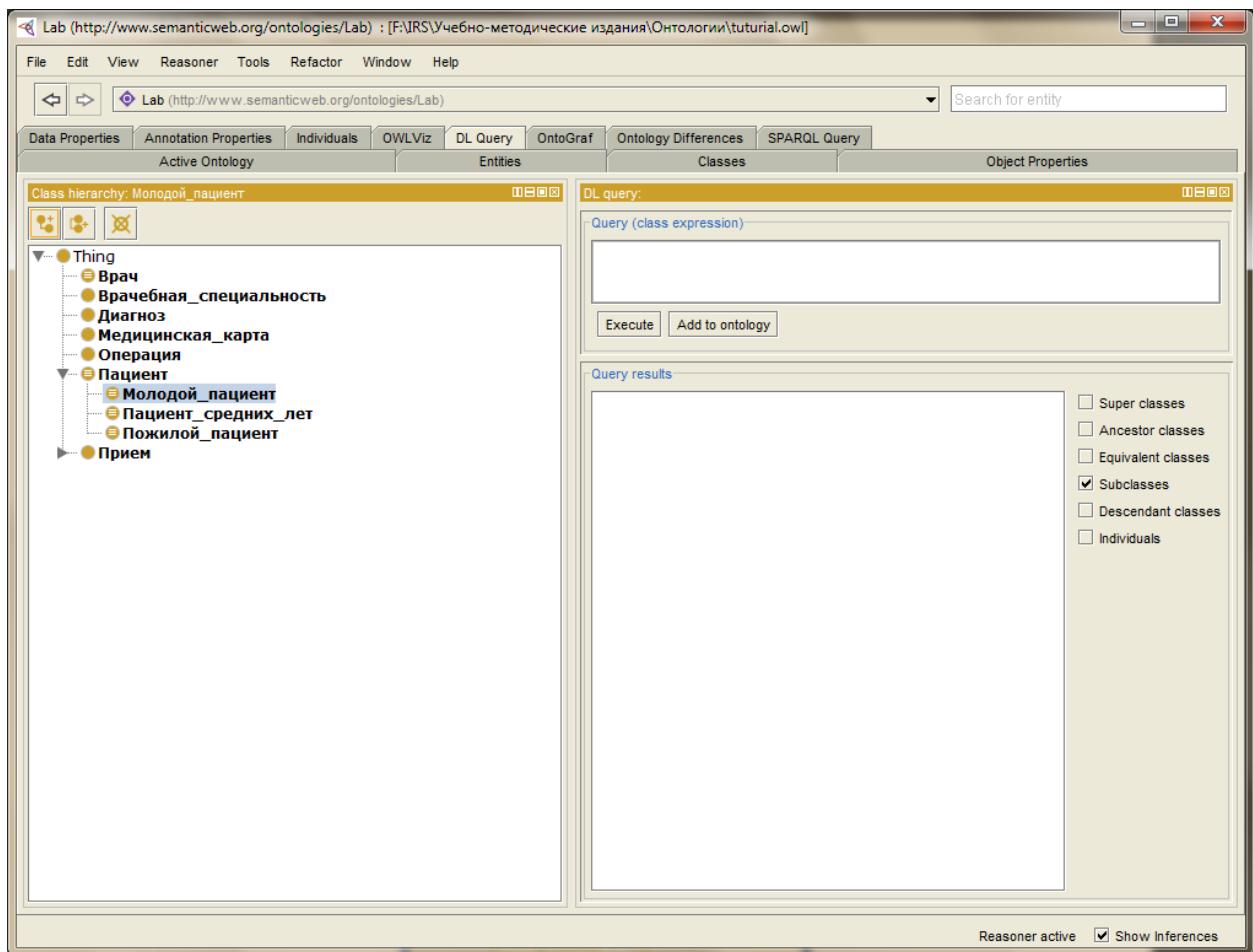


Рис. 5.39. Вкладка создания запросов («DL-Query»)

Базовые запросы позволяют вывести для выбранного класса его суперкласс, предков, эквивалентные классы, подклассы, потомков, экземпляры. Чтобы создать необходимый запрос, нужно выбрать класс, в котором будет производиться поиск, свойство, по которому будет производится поиск, а также указать признак. Под признаком может пониматься как строчка, так и условия is, is not, contains, begins with и так далее. Созданный запрос можно сохранить в онтологии, нажав кнопку «Add to ontology».

К примеру, мы хотим вывести все экземпляры класса «Пациент». Для этого в запросное окне необходимо перетащить название класса и выбрать на панели результатов «Individuals» (рис. 5.40).

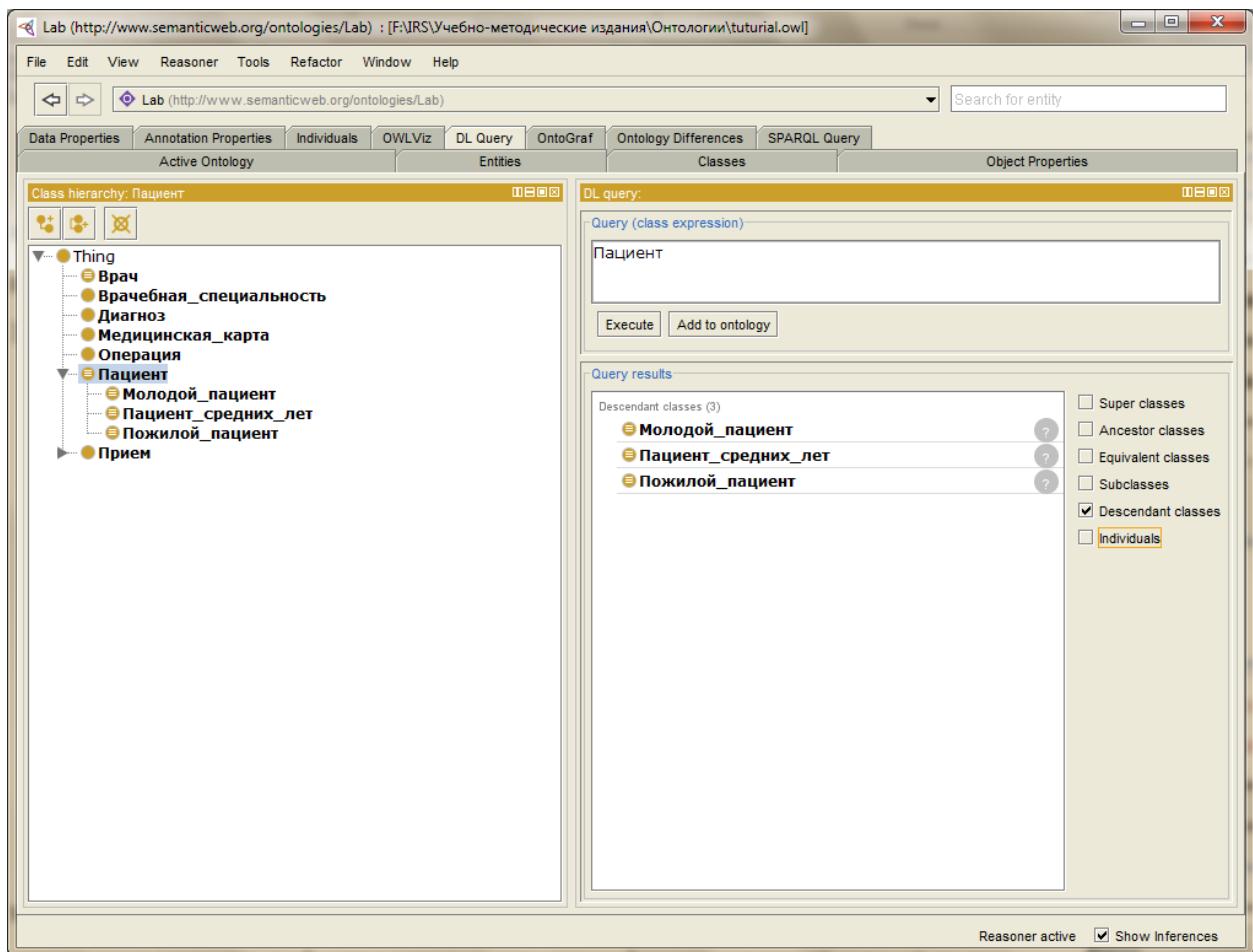


Рис. 5.40. Результат запроса

Аналогично можно создать запросы к онтологии, позволяющие вывести для выбранного класса его суперкласс, предков, эквивалентные классы, подклассы, потомков, экземпляры. Также при построении запросов можно пользоваться построителем выражений.

ЛИТЕРАТУРА

1. Antoniou G., Frank van Harmelen. Web Ontology Language: OWL. (in Handbook on Ontologies). – Springer Verlag, 2004. – P. 67 – 92.
2. Baader F. (editor), et al. The Description Logic Handbook. – Cambridge University Press, 2003. – 574 p.
3. Berners-Lee T., Hendler J., Lassila O. The Semantic Web // Scientific American. – May 2001.
4. Despres C., Chauvel D. The Present and the Promise of Knowledge Management. – Butterworth-Heinemann, 2000. – 352 p.
5. Handschuh S., Staab S. (eds.). Annotation for the Semantic Web. Volume 96 Frontiers in Artificial Intelligence and Applications. – IOS Press, 2003. – 240 p.
6. Maier R. Knowledge management systems: Information and communication technologies for knowledge management. – Berlin Heidelberg: Springer Verlag, 2004. – 635 p.
7. Антониоу Г., Грос П., Хармelen ван Ф., Хоекстра Р. Семантический веб. – М. : ДМК Пресс, 2016. — 240 с.
8. Букович У., Уильямс Р. Управление знаниями: руководство к действию (Wendi R. Bukowitz, Ruth L. Williams The Knowledge Management Field- book). – М.: ИНФРА-М, 2002. – 504 с.
9. Вебер А.В., Данилов А.Д., Шифрин С.И. Knowledge-технологии в консалтинге и управлении предприятием. – М.: Наука и техника, 2002. – 176 с.
10. Гаврилова Т.А., Хорошевский В.М. Базы знаний интеллектуальных систем. – Спб: Питер, 2000. – 384 с.
11. Люггер Д.Ф. Искусственный интеллект: стратегии и методы решения сложных проблем. 4-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 864 с.

- 12.Маннинг К. Д., Рагхаван П., Шютце Х. Введение в информационный поиск. – М. : Вильямс, 2011. – 528 стр.
- 13.Осипов Г.С. Методы искусственного интеллекта. – М.: ФИЗМАТЛИТ, 2011. – 296 с.
- 14.Палагин А.В., Крывый С.Л., Петренко Н.Г. Онтологические методы и средства обработки предметных знаний: монография /. – Луганск: изд-во ВНУ им. В. Даля, 2012. – 324 с.
- 15.Тузовский А.Ф., Чириков С.В., Ямпольский В.З. Системы управления знаниями (методы и технологии) / Под общ. ред. В.З. Ямпольского. – Томск: Изд-во НТЛ, 2005. – 260 с.
- 16.Цуканова Н.И. Онтологическая модель представления и организации знаний. Учебное пособие для вузов. – М.: Горячая линия–Телеком, 2016. – 276 с.

Оглавление

Глава 1. Представление знаний в интеллектуальных информационных системах.....	2
1.1. Данные и знания.....	2
1.2. Концептуальная парадигма работы со знаниями	4
1.3. Представление знаний с использованием формальных логических систем	9
1.4. Сетевые модели представления знаний.....	10
1.5. Продукционные модели представления знаний	12
1.6. Фреймовые модели представления знаний	14
Глава 2 Модели и методы онтологического инжиниринга.....	17
2.1. Предпосылки появления онтологической модели представления знаний и перспективы использования.....	17
2.2. Формальная модель онтологии.....	21
2.3. Типы онтологий. Формальная модель онтологической системы	24
2.4. Характеристики качества онтологий интеллектуальных информационных систем.....	28
2.5. Жизненный цикл создания онтологии	31
2.6. Особенности проектирования предметных онтологий	43
2.7. Отображение онтологий	46
2.8. Онтологии как основа семантического веба	50
Глава 3. Дескриптивные логики как формальные модели онтологий... 	58
3.1. Базовые формализмы дескрипционных логиках	58
3.2. Синтаксис и семантика логики ALC.....	60
3.3. Терминология логики ALC	63
3.4. Система фактов логики ALC.....	66
3.5. База знаний логики ALC.....	67
3.6. Разрешимость дескрипционной логики. Понятие разрешающего алгоритма	69
3.7. Табло-алгоритм для логики ALC.....	72

Глава 4. Языки описания онтологий	76
4.1. Виды языков описания онтологий	76
4.1. Язык OWL и его виды. Связь дескрипционных логик с OWL.....	79
4.2. Структура документа OWL.....	81
4.3. Описание классов OWL.....	83
4.4. Аксиомы классов.....	91
4.5. Свойства OWL.....	94
4.6. Описание экземпляров классов в OWL	98
4.5. Профили языка OWL	101
ГЛАВА 5. Онтологический инжиниринг в системе Protege.....	104
5.1. Программный инструментарий для онтологического инжиниринга 104	
5.1. Создание нового проекта в PROTEGE.....	109
5.2. Создание комментариев к онтологии.....	112
5.3. Сохранение проекта.....	114
5.4. Способы описания классов	115
5.5. Создание именованного класса	116
5.6. Использование мастера создания иерархии классов.....	124
5.7. Свойства онтологии	128
5.8. Создание свойства объектов	130
5.9. Виды ограничений	134
5.10. Создание кванторных ограничений	135
5.11. Использование машины вывода для проверки непротиворечивости иерархии классов.....	139
5.12. Описание ограничений мощности.....	142
5.13. Свойства типа данных	144
5.14. Описание ограничений на свойства типа данных	148
5.15. Создание DL-QUERY запросов	149
Литература.....	152