СЕРГЕЙ ГОРШКОВ

ВВЕДЕНИЕ В

ОНТОЛОГИЧЕСКОЕ МОДЕЛИРОВАНИЕ

тринидата | 2016

Сергей Горшков
ООО «ТриниДата»

Введение в онтологическое моделирование

ревизия 2.2

http://trinidata.ru
serge@trinidata.ru

Методическое пособие предназначено для аналитиков, ИТ-специалистов, менеджеров, участвующих в создании информационных систем, использующих семантические технологии для решения прикладных задач, студентов соответствующих специальностей.

Работа с пособием не требует специальной подготовки, за исключением общей эрудиции в ИТ. Хотя онтологическое моделирование и стек семантических технологий построены на математических теориях, таких как логика первого и второго порядка, в изложении мы намеренно не используем их формализм и обозначения. Необходимый минимум понятий дается в тексте пособия.

Пособие содержит достаточный материал для того, чтобы осознать возможности и ограничения семантических технологий, научиться проектировать и оценивать создаваемые с их помощью программные решения, находить источники информации для дальнейшего профессионального развития в этой области.

Пользуясь случаем, хочу поблагодарить людей, сыгравших огромную роль в наших работах и исследованиях: Сергея Гумерова, Евгения Свалова, Олега Рагозина, Максима Мирошниченко. Во многом благодаря им увидели свет и наши программные продукты, и это пособие.

Содержание

1. Введение

1.1. Компьютер и мыслительные задачи

Писателям-фантастам XX века казалось, что развитие вычислительных машин приведет к появлению интеллектуальных помощников человека, которые будут решать за него многие мыслительные задачи. Технические характеристики сегодняшней аппаратуры превышают самые смелые прогнозы многих из этих авторов: компьютер умещается на ладони, всемирная сеть доступна практически везде. При этом для решения аналитических задач мы в большинстве случаев по-прежнему пользуемся в лучшем случае электронными таблицами вроде Excel. Это особенно заметно в бизнес-среде, где цена (не)правильно принятого решения имеет совершенно осязаемый эквивалент в виде многомиллиардных прибылей или убытков. Тем не менее, развитие информационной инфраструктуры бизнеса завязло на пути создания крупных «трехбуквенных систем» (ERP, CRM и т.д.), на которые тратятся огромные средства, но которые не способны дать организации-владельцу ничего особенно «интеллектуального». Рассмотрим некоторые распространенные подходы, применяемые для решения бизнес-задач, формально относящихся к категории «интеллектуальных»: это позволит обосновать необходимость использования новых подходов, требования к используемым технологиям.

Современные системы «бизнес-аналитики» (BI) в основном заняты вычислением значений количественных показателей, часто имеющих весьма слабое отношение к описанию реальности, и манипулированию ими. Отличным примером служит любимый бизнесом показатель EBITDA: он характеризует прибыль, и по этой причине часто используется, например, в качестве базы для начисления бонусов топ-менеджерам. Однако он не характеризует эффективность работы менеджера в том смысле, в каком ее интуитивно оценивает собственник: ведь путем уменьшения расходов можно увеличить значение EBITDA. Это всегда интересно менеджеру, но не всегда оправданно с точки зрения стратегического развития предприятия. А уж при расчете этого показателя по подразделениям компании

возможности манипуляции открываются широчайшие. В большинство статей доходов и расходов вносят вклад сразу несколько подразделений, настройкой алгоритма расчета можно легко «награждать» фаворитов и «наказывать» неугодных. Разумеется, подобные маневры не имеют ничего общего с достижением реальной эффективности работы предприятия.

Еще рельефнее видны методологические проблемы при попытках решать оптимизационные задачи количественными методами. Типичный подход к этому вопросу состоит в формулировании «целевой функции», которая представляет собой описание какого-либо качественного состояния системы, представленного в виде числа – например, «обеспеченность населения такими-то услугами». Далее, также в количественной форме задаются ограничения, варьируемые параметры, и после вычислений получается некий набор «оптимальных» решений. Однако их практическое воплощение часто приводит к результатам, противоположным поставленным целям, или имеет серьезные побочные последствия. Например, легко может оказаться, что «средняя температура по больнице» – обеспеченность услугами – достигла нужных значений, но определенным группам населения они стали вовсе недоступны. Или же качество этих услуг снизилось настолько, что они практически потеряли смысл для потребителей. Легко понять, что корень проблемы лежит в

Модель всегда создается в интересах какого-либо экономического субъекта, и должна отвечать его интересам. В современной экономике интересы хозяйствующих субъектов и их собственников далеко не всегда связаны с получением прибыли от операционной деятельности предприятия. Классический пример – стартапы: сами по себе они чаще всего не приносят прибыли, однако их капитализация огромна. Задача владения таким предприятием – повышение его стоимости для того, чтобы продать как можно дороже следующему инвестору.

Мотивация собственников и менеджеров может быть еще менее связана с очевидными метриками эффективности бизнеса. В ней необходимо разобраться, для того чтобы решать задачу, действительно важную для пользователя модели.

слишком серьезных модельных допущениях, которые были сделаны при формализации целевого параметра.

Не менее рельефно недостатки распространенных методик моделирования проявляются в области построения моделей бизнес-процессов. Многие предназначенные для этой цели нотации и фреймворки априорно задают такой объем упрощений и ограничений, что реальное управление деятельностью предприятия на основе построенных моделей оказывается практически невозможным.

Вопросы для размышления

Как вы считаете, можно ли полностью возложить на информационную систему принятие решений, затрагивающих жизнь большого числа людей – или это прерогатива политиков? Есть ли задачи, которые принципиально не подлежат автоматизации?

Указанные методические проблемы напрямую связаны со способами автоматизированной обработки информации – точнее, с ограниченностью той их части, которую освоило бизнес-сообщество. Ведь если более сложный и достоверный алгоритм расчета какого-либо показателя или получения вывода нельзя, по мнению бизнес-заказчика или разработчика, реализовать в информационной системе – это обосновывает применение неверного, грубого, но технологически понятного способа расчета. Таким образом, в сущности, в сфере бизнеса человек пока по-настоящему доверил компьютеру только одну функцию – складывать и вычитать числа. Все остальное он по-прежнему делает сам, и делает, в большинстве случаев, не слишком качественно.

Разумеется, мы говорим только об общей тенденции; есть немало контрпримеров реализации по-настоящему эффективных систем, помогающих оптимизировать те или иные процессы, но почти все такие системы имеют узкую отраслевую направленность и содержат жестко запрограммированные алгоритмы решения задач. Таким образом, системного влияния на положение дел они не оказывают.

Что же нужно сделать для того, чтобы компьютер стал по-настоящему помогать нам в решении интеллектуальных бизнес-задач, смог поддерживать принятие решений в любых сферах? Необходимо вдохнуть в него «искру разума», то есть научить его решать задачи способом, подобным лучшему из тех, что используют люди. Фактически, для этого нужно воспроизвести в цифровом представлении те информационные структуры и процессы, которыми мы сами пользуемся в процессе мышления: понятийный аппарат, логические рассуждения. Тогда мы сможем реализовать и процессы обработки этих структур, то есть имитировать на компьютере отдельные фрагменты наших когнитивных способностей. После этого, получив определенные результаты, мы сможем критически посмотреть на смоделированные структуры и процессы, и улучшить их. В сочетании с недоступной человеку способностью вычислительных машин к быстрой обработке огромных объемов информации, такой подход обещает дать небывало высокий уровень качества поддержки принятия решений со стороны информационных систем.

Вопросы для размышления

Оправдано ли с этической точки зрения создание вычислительных машин и алгоритмов, способных выполнять мыслительные функции в полном смысле слова *лучше*, чем человек? Допустимо ли использовать интерфейсы нервная система – компьютер (уже созданные учеными) для дополнения возможностей человеческого разума? Допустимо ли объединять умы нескольких людей в «сеть» для более быстрого и качественного решения задач?

Это не фантастика и не праздный вопрос – например, ученые университета Дьюка давно проводят успешные эксперименты в этом направлении.

Мы не случайно привели именно логическое мышление в качестве примера когнитивного процесса, который можно воспроизвести в вычислительной среде. Существуют и другие подходы, наиболее популярным из которых является использование нейросетей – то есть имитация процессов, происходящих при взаимодействии нейронов в головном мозгу. При помощи такого рода средств

успешно решаются задачи распознавания образов, речи и т.д. Можно «обучить» нейросети и для применения в качестве средства поддержки принятия решений. Однако с ростом числа факторов, требуемых для оценки ситуации, сложности их структуры, способов влияния на ситуацию, возможности нейросетей становятся все менее убедительными: на обучение требуется больше времени, получаемые результаты носят вероятностный характер, не обеспечивают логической доказуемости, легко теряют воспроизводимость. Выход за пределы заранее ограниченного круга ситуаций приводит к невозможности получить от нейросети результат, пригодный для практического использования. Похожие недостатки имеют технологии машинного обучения. Имитация же логического мышления свободна от большинства этих недостатков, а коррекция логической схемы при изменении условий требует куда меньших усилий, чем переобучение нейросети. Зато при составлении логических моделей принципиально важным становится их корректность, непротиворечивость, релевантность, зависящая от человека – автора модели.

Можно ли построить при помощи только логических вычислений систему, способную претендовать на звание «искусственного интеллекта»? Ведь при помощи простых логических выводов из известных фактов, казалось бы, нельзя получить принципиально новой информации? Такое утверждение не вполне верно. Все развитие человеческой мысли происходило путем дедукции, получения умозаключений о новых для человека феноменах путем их уподобления уже известным явлениям. Дополнив имитацию строгого логического мышления инструментами, позволяющими делать выводы такого рода, можно получить самообучающуюся систему, способную к производству принципиально новой информации. Чтобы прийти к такой реализации, надо сначала получить целостную картину того, как знания представляются в электронном виде и обрабатываются при помощи уже существующих технологий – этому и посвящено наше пособие.

Одна из главных особенностей человеческого сознания состоит в том, что оно лениво. Наш мозг отсекает все «лишнее», сводя наше представление о событиях и явлениях к довольно простым определениям. Мы видим только черное и белое, и принимаем решения, исключив из рассмотрения подавляющее большинство объективной информации.

Этим же грехом человек страдает при анализе бизнес-процессов и сред. Вместо того, чтобы воспринимать бизнес как сложную систему, не поддающуюся упрощению дальше определенного предела без критической потери достоверности результатов аналитики, человек старается свести все критерии оценки и управления к нескольким числовым показателям. Таким образом удается упростить получаемую модель, снизить затраты на ее создание. Но поступающим так не следует удивляться, когда их прогнозы не оправдываются, а решения, принятые на основании моделирования оказываются неверными.

Главный принцип качественной аналитики, управления, основанного на знаниях, звучит так: **НЕ УПРОЩАТЬ** модель без необходимости.

1.2. Онтологическое моделирование: цели и средства

К сожалению, распространенные сегодня компьютерные технологии не благоприятствуют реализации этого принципа. Если в качестве инструмента анализа нам доступен только Excel или реляционные базы данных – описание бизнеса неизбежно придется сводить к ограниченному набору числовых показателей. Поэтому одной из наиболее актуальных проблем развития ИТ в настоящий момент является доведение до широкой промышленной эксплуатации таких технологий, которые позволяют строить действительно сложные и комплексные модели, и решать с их помощью те оптимизационные, аналитические, оперативные задачи, перед которыми другие технические средства оказываются бессильны.

Многообещающим, но несколько недооцененным на сегодняшний день направлением решения этой задачи является использование так называемых *семантических технологий*. Идеи автоматизированной обработки концептуализированного знания неоднократно выдвигались мыслителями начиная со Средних веков, актуализировались в Новое время, ограниченно использовались в лучшие годы советской плановой экономики, но до действительно функционального воплощения доросли только сейчас. На сегодняшний день созданы все необходимые компоненты методики и технологий, необходимых для работы с онтологическими моделями, которые являются предметом обработки с помощью семантических технологий. Слово «онтология» означает совокупность знаний; термин «семантические технологии» подчеркивает тот факт, что они обеспечивают работу со *смыслом* информации. Таким образом, переход с традиционных ИТ на семантические технологии является переходом от работы с *данными* к работе со *знаниями*. Разница между этими двумя терминами, которые здесь мы используем исключительно в применении к содержанию информационных систем, подчеркивает отличие в способе использования информации: для восприятия и

Идею «семантической паутины», которая должна стать следующим шагом в развитии всемирной сети, предложил Тим Бернерс-Ли, один из авторов концепции и технологий Интернета. Он считал, что узлы сети могут предоставлять информацию не только в виде текста или картинок, но и в формализованном представлении, которое позволит алгоритмам обрабатывать смысл информации. Появилась бы возможность связывать воедино информацию об одних и тех же объектах, представленную на разных узлах, использовать не текстовый, а логический поиск. Вместо «свалки» не систематизированного контента Интернет бы превратился в гигантский глобальный граф. В первоначальном виде эта смелая идея является, по-видимому, утопичной, хотя время только подтверждает необходимость в создании чего-то подобного. Технологии, созданные для ее реализации, очень пригодились для решения других задач.

использования *данных* человеку необходимо выполнять интерпретацию, выявление их смысла и его перенос на интересующую часть реальности (в тех случаях, когда это делает программный алгоритм – ситуация принципиально не меняется, так как способ интерпретации данных все равно задан человеком). Представленные же в электронной форме *знания* могут восприниматься непосредственно, так как они уже выражены при помощи того понятийного аппарата, которым пользуется человек. Кроме того, с такими знаниями (онтологиями) могут выполняться и полностью автоматические операции – получение логических выводов. Результатом этого процесса станут новые знания.

Аналитики Gartner называли семантические технологии одним из наиболее многообещающих ИТ-трендов 2013 года, однако их оптимизм оказался преждевременным. Почему? Все по той же причине – человек ленив, а создание семантических моделей требует серьезных умственных усилий. Тем больше выгод и преимуществ перед конкурентами получат те, кто предпримет эти усилия, и трансформирует их в реальный бизнес-результат.

Вопрос для размышления

Приведите пример, когда оптимальной стратегией для владельца предприятия является не управление, основанное на знаниях, а напротив – избежание любой интеллектуальной деятельности.

Таким образом, задачей онтологического моделирования является создание формализованных электронных моделей знаний. Цели применения этих моделей лежат в сфере бизнеса, и могут включать:

- Выполнение имитационного моделирования процессов с целью их оптимизации;
- Быстрое получение логических выводов на основании большого количества информации, с целью поддержки принятия решений;
- Обеспечение доступности для восприятия пользователей больших объемов сложно структурированной информации, обмен знаниями между людьми;

- Решение ряда технических задач, прежде всего в области интеграции информационных систем.

Необходимо заметить, что принятие решений – это волевой акт субъекта, напрямую зависящий от его интересов. Именно поэтому, говоря об информационной системе, мы рассматриваем только *поддержку* принятия решений, которая заключается в наилучшем обеспечении человека информацией, требуемой для их осознанного принятия, а также предсказании последствий тех или иных вариантов решения для того, чтобы человек мог оценить их приемлемость с учетом ценностных установок.

Говоря об оптимизации, достижении оптимального результата мы всегда имеем в виду, что этот результат является таковым только с точки зрения определенного субъекта, который совпадает для нас с владельцем или пользователем информационной системы. Могут быть реализованы средства балансирования интересов разных субъектов, но только в том случае, если этот баланс сам по себе отвечает интересам владельца системы. Именно по этой причине мы в основном будем рассматривать примеры и ситуации из бизнес-контекста, где интересы субъектов предельно понятны и общеприняты.

По нашему мнению, ни одна информационная система не должна претендовать на замещение волевых актов людей или групп – таких, как принятие политических решений, выбор личной судьбы, формулирование или популяризацию ценностных установок.

1.3. Онтологические модели: путь к результату

Создание и использование онтологических моделей – междисциплинарный вид деятельности, который требует определенной эрудиции в информационных технологиях, математике и логике, философии, моделируемых предметных областях. Для достижения качественного результата создания и использования моделей необходимо проявить широту и логичность мышления на всех этапах – от постановки целей и проектирования решения, до формализации знания и

интерпретации полученных результатов. В связи с этим основной задачей нашего пособия является формирование у читателя как можно более широкой картины применения онтологического моделирования. Корректно построенная модель не принесет пользы без адекватной программной реализации; ни одна безупречно спроектированная ИТ-система не принесет пользы без четкого понимания целей и способа применения получаемых результатов. Таким образом, мы рекомендуем читателю позиционировать себя при чтении пособия не как аналитика, программиста или менеджера, а как человека, полностью ответственного за создание качественного методического и технического решения определенных проблем.

Структура пособия отвечает этой задаче. Оно состоит из двух «кругов» понимания: в главах 1-5 дается общее представление об используемых методиках и технологиях, а начиная с шестой главы происходит погружение в их детали. При этом основной объем фактических знаний все равно остается за пределами пособия – получить его можно самостоятельно при помощи доступных в Интернете материалов. Нашей задачей является скорее рассказ о том, что именно, как и для чего искать, нежели передача большого объема фактических знаний.

И, конечно же, важнейшей задачей является поддержка критического, логического мышления, «чистого» взгляда на мир, как можно меньше искаженного привычными концепциями с уже реализованным потенциалом, вроде реляционной модели данных или большинства нотаций моделирования бизнес-архитектур и процессов. Умение на определенных этапах отстраниться от технических средств решения задачи, сконцентрировавшись на концептуальных вопросах, а затем вернуться к ним и эффективно воплотить сформулированные концепции, критически важно для успеха моделирования.

Пожелаем читателю удачи на этом пути.

2. Знакомство с онтологическим моделированием

2.1. Как мы передаем и воспринимаем информацию

Перед тем, как начать рассказ о семантических технологиях, необходимо дать определения основных терминов, которые мы будем использовать.

Ключевым для нашего рассказа термином является *информация*.

Информация – это сведения о чем-либо, имеющие определенное физическое представление[1]. Физическое представление – это способ хранения информации на определенном носителе: она может быть записана на бумаге, храниться в клетках человеческого мозга или в памяти ЭВМ. Средствами физического представления информации являются определенные последовательности объектов или сигналов. Набор таких объектов или сигналов образует *информационное сообщение*

Практическая ценность информации обеспечивается возможностью ее передачи. В процессе обработки информации участвуют:

- Источник – объект или субъект, создающий информационное сообщение;
- Носитель – физическая среда, хранящая и/или переносящая сообщение;
- Приемник – объект или субъект, воспринимающий сообщение.

Для того, чтобы информационное сообщение обрело практическую ценность (*прагматику*), источнику и приемнику нужно согласовать между собой:

- Физическую форму представления информации;
- Правила формирования внутренней структуры сообщения (*синтаксис*);

[1] Общепринятого определения информации не существует; в разных предметных областях термин «информация» может нести разную предметную нагрузку. Приведенное нами определение близко по смыслу к тому, что дает словарь Ожегова. Мы здесь рассматриваем этот и другие термины только в том узком значении, которое необходимо для нашего рассказа.

- Способы выражения смысла сообщения (*семантику*).

В классической теории информации ее количество определяется как мера устранения неопределенности относительно чего-либо. Не вдаваясь в подробное рассмотрение этого вопроса, будем иметь в виду, что объем информации связан с количеством лишь косвенным образом: большой объем не означает наличия большого количества информации.

Понятие «*смысл* сообщения» обретает значение только относительно живого существа, которое формирует или воспринимает информационное сообщение. Для электронно-вычислительных машин сообщения никакого «смысла» не несут, даже если нам кажется, что компьютер осуществляет «сознательные» действия. Однако компьютеры могут помочь живым существам – людям – в обработке и восприятии смысла информации, в том числе в получении логических выводов, то есть продуцировании одной информации из другой.

Информация, имеющая смысл, образует *знания* живого существа. Знания – это та практически полезная информация, которую мы можем использовать в своей деятельности. Заметим, что мы не говорим об истинности (релевантности), или полноте знаний: в принципе, любые знания всегда не полны и совершены. Но для их обладателя они имеют непосредственный смысл и значение.

Вопросы для размышления

Животные тоже умеют получать знания и пользоваться ими. Есть ли принципиальное отличие между тем, как получают и используют знания животные и современные информационные системы? Какая граница тоньше – между человеком и животным, или живым существом и компьютером?

Информация поступает в наш мозг от органов чувств. При этом происходит многоступенчатый процесс *интерпретации*, в результате которого мозг выражает поступившую информацию при помощи совокупности *понятий*, или *концептов*.

Понятия представляют собой относительно устойчивые структуры мышления, используемые для объединения родственных явлений, придания им тех или иных свойств, выражения закономерностей, получения логических выводов на их основе. Понятия могут характеризоваться большим или меньшим уровнем абстрактности (то есть сложности связи понятия с объектами или явлениями реального мира), эмоциональной или ценностной оценки и т.д. Для нас важно то, что в процессе интерпретации информации, в ходе работы второй сигнальной системы (в физиологическом понимании), мозг конструирует многослойную структуру, которая представляет собой набор понятий и связей между ними. Эта структура необходима для речевой деятельности – формулирования и восприятия высказываний, для логического мышления, но может существовать и безотносительно нее. Именно такие структуры и составляют *знания* человека, на основе которых он осуществляет свою деятельность. Люди могут обмениваться знаниями при помощи множества средств – от устной речи до электронных сообщений. Для выполнения поставленных нами целей необходимо добиться того, чтобы информационные системы могли оперировать знаниями, а не только механически хранить и передавать их.

2.2. Принципы построения концептуальных и информационных моделей

В этом разделе и далее мы сосредоточимся на процессе создания онтологических моделей. Под онтологической моделью будем понимать концептуализированное представление информации о какой-либо области реальности, представленное в электронном виде. Таким образом, мы будем одновременно рассматривать процессы построения мысленной (концептуальной) модели, и ее воплощения в электронной форме в соответствии с методикой онтологического моделирования. Словом «концептуальный» мы хотим подчеркнуть тот факт, что модель строится при помощи понятийного аппарата.

Проследим процесс формирования модели на примере. Пусть человек смотрит на следующую картинку:

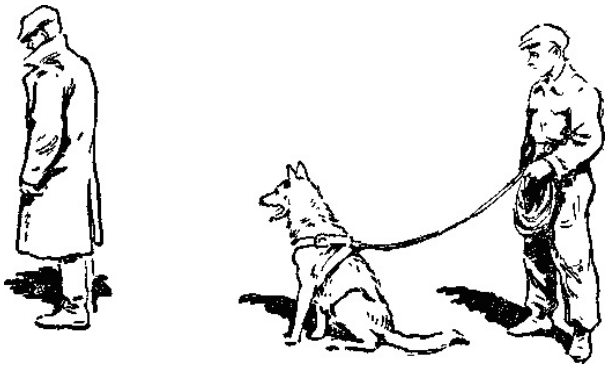

Рис. 1. Пример информационного сообщения

Первым делом человек выделяет основные объекты, представленные на изображении, и дает определения каждому из них: человек, собака, человек. Этот процесс можно назвать *концептуализацией*: для каждого объекта мы подобрали концепт (понятие), который соответствует ему в нашей картине мира. Таким образом, мы установили соответствие между уже имевшимся у нас понятием (знаком), и мысленным образом конкретного предмета (означаемым), возникшим у нас при восприятии изображения. При этом мы использовали наше внутреннее «определение» этого понятия – некую идею, скрывающуюся за ним. Такие идеи называются интенсионалами. Связь между предметом, его мысленным образом, интенсионалом и символов показана на следующем рисунке.

Зоолог Конрад Лоренц описал следующий эксперимент, ставший классическим: если над домашними птицами, прогуливающимися по двору, перемещать на шнуре фанерный силуэт ястреба – у них начинается паника из-за опасности. Если тот же силуэт перемещать хвостом вперед, то паники не возникает – одно из условий, по которому птица определяет опасность, не выполняется. Точно таким же образом работает человеческое восприятие, нацеленное на выявление знакомых паттернов в происходящем. Под контролем сознания восприятие протекает лишь в редких случаях.

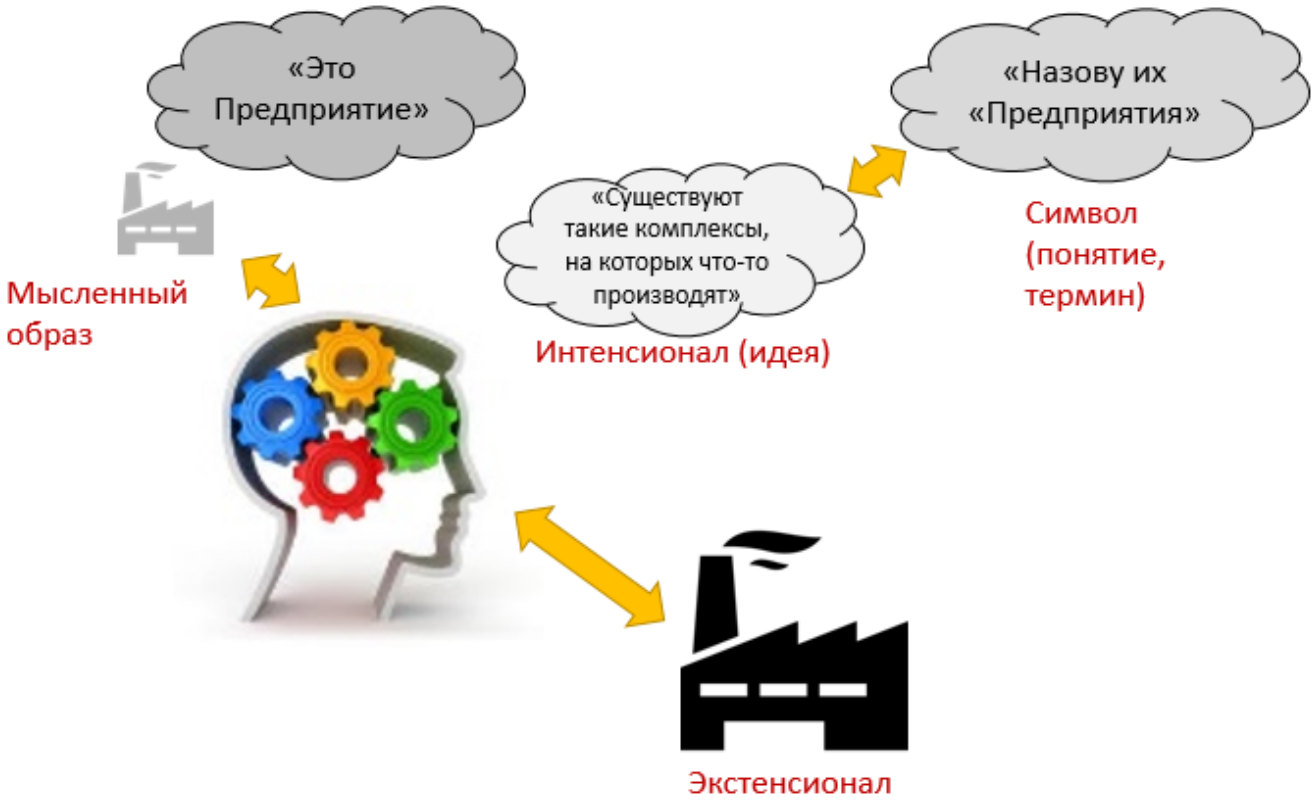

Рис. 2. Интенсионал, символ, экстенсионал и мысленный образ

Итак, при моделировании важно четко различать:

- объект или явление реального мира,
- его образ в нашем сознании,
- понятие (символ, знак), которое мы используем для обозначения этого образа,
- идею, или интенсионал, соответствующую понятию.

Понятия используются для группировки сущностей, однородных в определенном отношении, например – обладающих некоторым набором признаков. Так, собака для нас – четвероногое животное, имеющее определенную форму и пропорции тела, морды, ушей и т.д. Таким образом, можно сказать, что концепт «собака» соответствует *классу* объектов реального мира. Принадлежность конкретного объекта к классу позволяет нам автоматически наделить объект некоторыми свойствами, которые могут явно не содержаться в интерпретируемом сообщении: так, мы *знаем*, что собака может укусить, бегает быстрее человека и т.д.

Таким образом, в процессе восприятия ситуации человек использует уже имеющиеся у него знания для того, чтобы вписать смысл нового информационного сообщения в общую структуру имеющихся знаний. Благодаря этому, сообщение обретает субъективный (прошедший сквозь призму личного опыта) смысл, и образует новые фактографические знания субъекта.

Вернемся к процессу восприятия. Выделив и классифицировав объекты, человек оценивает отношения, в которых они находятся. Очевидно, что собака сторожит человека, изображенного на нашем рисунке слева: если он попробует сбежать, собака его задержит. На основании внешнего вида собаки мы отнесли ее к классу служебных собак, которые, как нам известно, способны на такие действия. Человек слева кажется нам похожим на подозреваемого в преступлении, а поза собаки и держащий ее на поводке кинолог подкрепляют сделанный вывод. На этом шаге мы перешли к более высокому уровню концептуализации: от предметных понятий (человек, собака) мы перешли к функциональным (служебная собака, подозреваемый). Развивая рассуждение, мы перейдем к еще более сложным понятиям, подразумевающим более высокий уровень абстракции (то есть сложности связей с объектами и событиями реального мира), таким как «преступник», «закон», «наказание». Особенность человеческого мышления состоит в том, что, используя подобные понятия, мы вешаем на объекты и явления определенные ярлыки, теряем непредвзятость и способность оценивать воспринимаемую картину в целом. Это может показаться печальным, но такая особенность является одним из приемов адаптации к окружающей среде – без этого человек просто утонул бы в бесконечном осознании множества воспринимаемых образов.

Вопрос для размышления

Как вы думаете, что является целью школьного образования – обучение человека самостоятельному мышлению, или формирование паттернов, которые обеспечат его нормальное функционирование в обществе в определенных рамках?

Вся воспринимаемая информация проходит также сквозь призму эмоциональной сферы человека, на состояние которой влияют как физиологические, так и психические факторы. В результате мысленная модель, сложившаяся у случайного наблюдателя при восприятии сцены, изображенной на рис. 1, может получиться примерно такой:

Рис. 3. Результат интерпретации информационного сообщения

Очевидно, однако, что на приведенном на рис. 1 изображении никаких сведений о «преступниках» и «подозреваемых» не содержится. Просто в процессе интерпретации содержания изображения мы задействовали уже существовавшие в нашей голове понятийные структуры и правила получения логических выводов, применив их для описания наблюдаемой ситуации.

Здесь нужно иметь в виду, что слово «логический» не означает «правильный». Правила, по которым человек строит выводы, могут быть некорректными как с формальной точки зрения, так и по существу.

В результате мы построили *мысленную модель* этой ситуации. **Модель** – это информационное представление какой-либо совокупности объектов и явлений реального мира, характеризующееся:

- Упрощением – в модели представлена информация не обо всех чертах рассматриваемого фрагмента реальности, а только о тех, которые существенны с прагматической точки зрения конкретного субъекта.
- Концептуализированностью – каждый объект, явление, свойство выражены при помощи понятий разного уровня абстрактности.
- Взаимосвязанностью – все элементы модели связаны между собой.
- Наличием логических правил взаимодействия элементов.
- Прогностическим потенциалом – мы можем выполнить мысленный эксперимент, внеся в модель те или иные параметры или события, и при помощи указанных выше логических правил сделать вывод о том, что произойдет в том или ином случае (например, если человек попытается сбежать – кинолог спустит собаку, и собака задержит подозреваемого).

Прогностический потенциал и обеспечивает прагматику (практический смысл) создания модели. Однако очевидно, что степень достоверности выводов, которые мы можем получить в ходе мысленного эксперимента на модели, напрямую зависит от того, насколько качественной будет эта модель. «Качество» модели – многогранное понятие, включающее как степень достоверности и правомерности допущений, сделанных нами при построении модели, так и корректность самого понятийного аппарата. Для глубокого знакомства с этой темой рекомендуем следующие книги:

1. Sowa J. F. Knowledge representation. Logical, philosophical and computational foundations. Brooks/Cole, 2000
2. Partridge, Chris. Business Objects: Re-Engineering for Re-Use [2nd Edition], BORO Centre, 2005
3. West Matthew. Developing High Quality Data Models. Elsevier, 2011

К сожалению, на русский язык ни одна из этих книг не переведена.

Отметим, что приводимые в классических учебниках подходы при практическом применении следует оценивать еще и с экономической точки зрения: более «правильная» информационная модель может потребовать слишком больших

затрат на создание, поддержку, получение логических выводов. Задача оптимальной организации процесса моделирования состоит в том, чтобы обеспечить приемлемый уровень достоверности результата при минимальных затратах на его достижение.

Задача компьютерных технологий состоит в том, чтобы облегчить человеку процесс мышления. Значит, мы должны чуть более детально рассмотреть структуру таких процессов, чтобы понять, как можно их частично реализовать в электронном виде. Таким образом, мы перейдем от рассмотрения мысленного моделирования к моделированию средствами информационных технологий. При этом нас будет интересовать целенаправленное построение моделей, предназначенных для решения тех или иных практических задач, и одна определенный набор методик формализации таких моделей, который можно воплотить при помощи существующего сегодня стека семантических технологий.

2.3. Выделение объектов

Первый из когнитивных процессов, необходимых для построения информационной модели какой-либо области реальности, можно назвать *декомпозицией*. Он состоит в разделении моделируемого фрагмента реальности на отдельные элементы, которые станут базовыми единицами информационной модели. Во введенной нами терминологии выделяемые элементы реальности называются экстенсионалами, а наши представления о них – мысленными образами. В компьютерной модели мы будем отражать именно мысленные образы. Помня об этой разнице, далее для простоты изложения будем называть такие единичные элементы модели просто *объектами*.

В приведенном выше примере мы обошлись тремя объектами; однако, декомпозицию можно продолжить – выделить в качестве отдельных объектов поводок, детали одежды людей и т.д. Обратим внимание на то, что при первом

взгляде мы оценили человека, изображенного на картинке, как целостный объект – вместе с одеждой и обувью. Если он побежит, одежда будет перемещаться вместе с ним, поэтому на простейшем уровне моделирования такое обобщение допустимо. Однако если мы захотим спрогнозировать возможные действия этого человека более детально, декомпозицию нужно будет продолжить: например, человек может снять кепку или перчатку и бросить собаке, чтобы попытаться отвлечь ее на секунду. Состав объекта может быть нам до конца не известен: а вдруг у человека за пазухой спрятан пистолет? Тогда результаты нашего мысленного эксперимента существенно изменятся, события могут пойти по совсем другому сценарию. Это вопрос, в том числе, целостности полученного нами информационного сообщения, достаточности информации для моделирования.

При декомпозиции необходимо учитывать и возможную нестационарность существования объектов. Если собака сильно дернется и разорвет поводок, вместо одного объекта появятся два новых. С другой стороны, если собака откусит палец человеку, появится новый отдельный объект (палец), но и старый не прекратит существования (человек). Кстати, физическая неразрывность далеко не всегда служит основанием для логического объединения или разделения объектов. Кроме того, далеко не все объекты являются физически неразрывными, или вообще имеют однозначное материальное воплощение.

Глубина декомпозиции, определение границ объектов, а также решение о том, создавать ли (удалять ли) объекты в модели при существенном изменении их состояния, зависят только от прагматики – практического назначения модели. Если мы рассматриваем нашу картинку в целях отработки действий правоохранителей при задержании – то, конечно, подозреваемый останется подозреваемым независимо от того, какие повреждения будут ему нанесены. Если одежда или иные находящиеся при нем предметы могут быть им использованы для изменения хода событий (влияния на результат задержания) – мы должны включить эти предметы в модель как отдельные сущности. Также понятно, что если мы будем детально моделировать все имеющиеся у подозреваемого предметы, то составление и

интерпретация модели потребуют слишком много времени и ресурсов, что нецелесообразно с экономической точки зрения. Задача *правильного* моделирования состоит в поиске *практически обоснованного и оптимального* соотношения между уровнем детализации модели (и, как следствие, достоверности результатов моделирования) и требуемых для этого ресурсов.

Каждый выделенный нами в процессе декомпозиции объект мы будем называть *индивидуальным объектом*, или *индивидом* – это термин из стандарта OWL, с которым мы познакомимся далее.

Чтобы выразить сведения об объектах, которые позволят делать логические выводы на основании нашей модели, констатации существования самих объектов недостаточно. В модели будут присутствовать элементы как минимум трех других видов – различные виды группировки объектов, описания их свойств и связей. Подробно мы их обсудим далее, пока лишь констатируем, что целостная концептуальная модель, построенная по рассматриваемой нами методологии, будет содержать сущности таких типов:

- Индивидуальные (конкретные) объекты;
- Определения классов (групп) объектов;
- Определения статических атрибутов объектов;
- Определения связей между объектами.

Перечисленные виды сущностей описывают «терминологию» модели. Их описание можно уподобить заданию структуры информации в реляционной базе данных. В онтологическом моделировании эта часть модели называется TBox – от английского Terminology Box, набор терминов.

В модели, решающей конкретные практические задачи, будут также содержаться значения конкретных свойств для конкретных индивидуальных объектов, конкретные связи между ними. Эта часть модели описывает конкретные факты и называется ABox – от Assertion Box, набор утверждений. Третьей частью полной онтологической модели может быть набор правил получения логических выводов.

2.4. Идентификация объектов

Второй шаг в построении концептуальной модели назовем *идентификацией объектов*. Для того, чтобы думать о каком-то предмете, мы должны его идентифицировать, то есть однозначно обозначить. В общественной жизни идентификаторами служат имена людей, в географии – названия городов и стран, и так далее. Мы не можем осознать процесса идентификации безымянных объектов, таких, как собака и люди на нашей картинке, но, несомненно, в нашем мозгу эти идентификаторы существуют – просто этот процесс является для нас слишком «низкоуровневым» и не поддающимся осмыслению. Итак, первый шаг в составлении модели любой области реальности состоит в ее разделении на объекты, а второй – в идентификации этих объектов.

Когда мы составляем модель с практическими целями, очень важно соблюсти принцип нейтральности идентификаторов. Идентификатор сам по себе не должен нести никакого смысла, не имеет права основываться ни на каких

В случае с человеком границы его существования более или менее понятны. Теперь представим себе автомобиль, у которого за 20 лет службы заменили и кузов, и двигатель. К концу службы в нем не осталось ни одной детали из тех, что были установлены изначально, сменились все номерные агрегаты. Этот тот же автомобиль, или другой? Если другой, то где граница перехода?

Одно из уральских предприятий было основано в XIX веке как золотопромышленная фабрика, сменило нескольких владельцев, названий и видов деятельности. В годы войны на его территории разместили несколько эвакуированных заводов, которые постепенно влились в общую производственную структуру. Что считать объектом «завод» - площадку, оборудование, коллективы, юридические лица? Мы говорим об одном заводе, или нескольких? Как их разграничить?

Решение зависит от задач использования модели. Удобно разделить завод на разные объекты – активы, коллектив – и комбинировать их между собой в различные сущности, существующие в разные периоды времени.

свойствах объекта. Хорошо известно, что имена людей как идентификаторы – ненадежны: можно сменить имя, фамилию, а также пол, любые другие свойства человека, включая биометрические (при определенных обстоятельствах), человек же при этом останется тем же самым. Поэтому категорически недопустимо придавать идентификатору какой-либо смысл. Кстати, таким недостатком грешат практически все каталоги промышленной продукции. В идентификатор подшипника может входить информация о его внутреннем и наружном диаметре, типе, материале и т.д.; именно поэтому такие каталоги неизбежно устаревают, оказываются недостаточными для представления информации о новых типах изделий, обладающих иным набором свойств.

Идентификаторы имеет большинство сущностей, входящих в состав информационной модели: индивидуальные объекты, определения классов и определения свойств. Однако, эти сущности могут быть и «анонимными», о чем мы расскажем далее.

2.5. Классификация

Третьим шагом формального описания модели обычно является *классификация* – включение каждого объекта в один или несколько классов. Разумеется, чтобы приступить к классификации, необходимо сначала определить сам набор доступных нам классов.

Термин «*класс*» в применении к рассматриваемым в нашей книге информационным моделям означает некий символ (информационный сигнал), который используется для общего обозначения какой-либо совокупности объектов:

Рис. 4. Класс как совокупность индивидуальных объектов

Сам класс соответствует интенсионалу (идее), а его идентификатор или название – символу. При помощи этого символа мы обозначаем группу неких индивидуальных объектов, отвечающих идее, или совокупность других понятий (абстрактная идея). В простейшее определение класса не входят ни признаки, по которым мы относим к нему объекты, ни его отношения с другими классами, ни общие характеристики, которыми обладают все члены класса. Хотя вся эта информация может быть отражена в модели, например – в виде правил отнесения объектов к классам.

Один и тот же знак может иметь разные интенсионалы – по-разному интерпретироваться в зависимости от субъекта и контекста. Например, знак «победитель при Бородино» для русского будет соответствовать Кутузову, а для француза – Наполеону. На самом деле, у знака просто нет «объективного», самостоятельного смысла или значения – им его наделяет только субъект.

Важно понять, что за определением класса не таится никакой «объективной», «всеобщей» идеи. Аналитик может использовать в качестве названия класса термин

из обычного языка предметной области, наделив его такой прагматикой, которая оправдана в данной конкретной модели.

В самом деле, будет ли относиться к классу «собаки» фарфоровая собачка? Это целиком и полностью зависит от субъективной точки зрения, от прагматики, то есть от того, как именно мы будем использовать нашу модель. В модели, предназначенной для отработки действий при задержании подозреваемых – не будет. В модели, которую мы строим для выбора подарка на день рождения ребенку – собака может быть как живой, так и фарфоровой, и плюшевой. Таким образом, экстенсионал (именуемое, объем понятия, набор соответствующих ему объектов) одного и того же интенсионала (самого знака, понятия, принципа выделения группы объектов) варьируется в зависимости от контекста и субъекта, который его использует.

Концептуализация в мысленных моделях, и классификация – в информационных нужны для того, чтобы получить возможность формулировать знания и строить логические выводы. Для этого нужно включить в модель утверждения о классах, соответствующие арсеналу логики первого порядка. Одним из видов таких утверждений являются выражения формата «Все члены класса X являются членами класса Y» (например, «все собаки – животные»). С правилами такого рода мы познакомимся далее.

При помощи таких утверждений можно построить иерархию классов, от общих к более специфичным. Таким образом, классы образуют иерархию, или *таксономию*. Человек склонен максимально упрощать решение своих мыслительных задач. Поэтому естественной для нас организацией классов является простая иерархия – «дерево», в котором у каждого элемента есть только один родитель. Это ограничение естественно для людей (в особенности – для программистов), однако нужно понимать, что оно искусственно, и не имеет прагматических оснований. Практически полезной может оказаться не

нормализованная иерархия, где любой класс может имеет два и более не пересекающихся надкласса.

Поскольку составление классификаторов является важной частью практического моделирования, остановимся на этой теме чуть подробнее. Еще одним распространенным заблуждением составителей классификаций является комбинирование в одной иерархии разных оснований разграничения классов. Представим, что мы строим классификацию товаров магазина. При этом мы ограничиваем свой мыслительный процесс тем, что в складской программе, куда в конечном счете попадет классификатор, справочник товаров представляет собой одну иерархию, притом нормализованную. Это заставляет нас строить деревья такого вида:

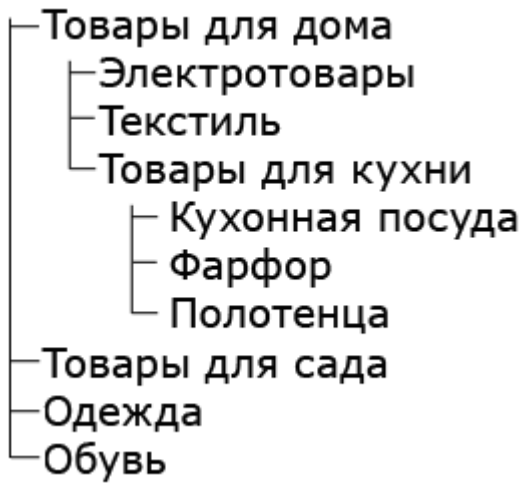

Рис. 5. Неправильно построенная иерархия

Из чего исходил аналитик при построении такой иерархии? Скорее всего, он следовал структуре отделов магазина. Верхний уровень иерархии соответствует функциональному делению товаров, поскольку люди чаще всего заходят в магазин с какой-то определенной целью (например, по дороге на дачу). Второй уровень отчасти отражает технологию производства товаров, отчасти – продолжает их функциональное деление. Наконец, третий уровень соответствует тому, как легче сгруппировать товар на полках.

Такая иерархия может показаться удобной для навигации по реальному магазину, но в информационной системе, предназначенной для помощи в выборе нужного товара (предположим, в Интернет-магазине) – она окажется крайне неудачной. Легко заметить, что, например, кухонное полотенце окажется в разделе «Товары для кухни» -> «Полотенца», хотя по технологическому признаку могло бы оказаться в разделе «Текстиль».

Чтобы избежать подобных трудностей, необходимо разобраться с основаниями классификации, а затем построить одну или несколько иерархий классов. Каждый индивидуальный объект мы сможем отнести одновременно к нескольким классам. Такой принцип классификации называется *фасетным*.

Вопрос для размышления

Придумайте основания классификации для классификатора отелей в каталоге турфирмы.

Заметим, что основания классификации сами по себе могут образовывать довольно разветвленные иерархии. На самом верхнем уровне их можно разделить на те, что описывают внутренние свойства объекта (материал, форма), и те, что характеризуют виды отношений, в которые объект может вступать с другими объектами (назначение, роль).

Проиллюстрируем сказанное на примере. Пусть мы хотим построить иерархию классов для товаров крупного магазина, и определить место в этой иерархии для конкретного товара – электродрели. Используем следующие основания классификации:

- Область применения товара;
- Способ использования;
- Функциональное назначение.

Заметим, что следует по возможности избегать таких именований для оснований классификации, как «вид» и «тип»: эти слова сами по себе не несут никакой информации, но скрывают за собой какие-то иные основания, для которых

мы просто не нашли подходящего определения. По области применения мы можем разделить товары так:

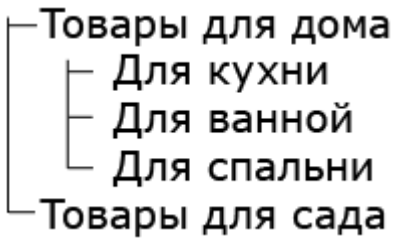

Рис. 6. Классификация по области применения

По способу использования:

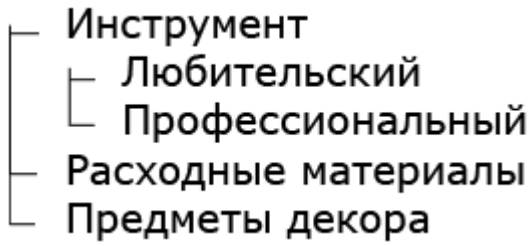

Рис. 7. Классификация по способу использования

Отметим, что классификация инструмента может основываться на множестве разных факторов; для нашей цели – построения навигационной системы магазина – можно разделить его по предполагаемым пользователям, что и сделано на рис. 7, а можно – по каким-то иным основаниям, например на режущий, сверлильный и т.д.

Наконец, по функциональному назначению классифицируем товары так:

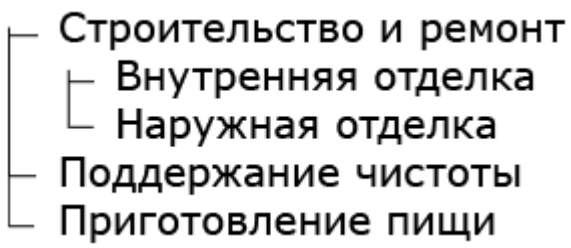

Рис. 8. Классификация по функциональному назначению

Разумеется, в каждом случае мы показали лишь небольшую часть иерархии товаров. Подчеркнем, что показанные способы классификации не являются «правильными» или «предпочтительными» сами по себе – они всего лишь могут быть рациональны в свете каких-то конкретных задач. Если изменятся цели, другой должна стать и классификация. В завершение этого примера, посмотрим на то, к каким классам будет отнесен товар «электродрель XYZ»:

Рис. 9. Классификация электродрели

Согласно этой классификации, электродрель модели XYZ – любительский инструмент, который может быть использован как дома, так и в саду, при работах по внутренней или наружной отделке помещений.

Как мы видим, один и тот же индивидуальный объект может быть отнесен сразу к нескольким классам, причем даже в пределах одной иерархии («товары для дома» и «товары для сада»).

У внимательного читателя, наверное, уже возник вопрос – почему мы называем электродрель XYZ индивидуальным объектом? Ведь если XYZ – это обозначение модели, то «Электродрель XYZ» должна быть классом, в который входят конкретные электродрели, каждая со своим серийным номером.

На самом деле, разделение на классы и индивидуальные объекты также зависит от контекста использования модели. Когда мы строим продуктовый каталог, ничто не мешает нам считать «Электродрель XYZ» индивидуальным объектом – в процессах продажи и складского движения товара нас не интересуют его конкретные единицы. Разумеется, это справедливо только в том случае, если мы

совершенно уверены, что нам не придется использовать ту же модель для описания других бизнес-процессов, где отслеживание индивидуальных единиц товара станет актуальным. В контексте бизнес-процесса сервисной службы, которая имеет дело с ремонтом дрелей, «Электродрель XYZ», безусловно, будет классом, к которому относятся конкретные устройства. Забегая вперед, скажем, что есть возможность считать одну и ту же сущность и классом, и индивидуальным объектом в пределах одной модели.

Еще один важный момент связан с глубиной детализации классификаций. Например, мы можем классифицировать собаку как животное, а можем построить «матрешку» с участием всех элементов биологической классификации: Животные – Хордовые – Млекопитающие – Хищные – Псовые – Волки (род) – Волк (вид) – Собака (подвид). Но будет ли такая классификация иметь практический смысл? В подавляющем большинстве случаев – нет, хотя может и иметь, например, если мы строим модель для генетических или медицинских исследований. Таким образом, при построении классификаций важно придерживаться принципа экономности, иначе говоря – отсекать при помощи «бритвы Оккама» все сущности и смыслы, наличие которых не оправдано прагматикой.

В следующих разделах мы расскажем о том, какие виды отношений можно задавать между классами, каким образом формулировать правила отнесения объектов к тем или иным классам. Такие правила выражаются при помощи логических утверждений («ни одна собака не является кошкой»), и используются в процессе получения логических выводов («Дружок – собака; значит, он не кошка»).

2.6. Описание свойств

Модель, состоящая только из классов и индивидуальных объектов, имеет мало практического смысла. Мысленно оперируя объектами, мы приписываем им

те или иные *свойства,* причем правила взаимодействия объектов почти всегда зависят от значений этих свойств. Таким образом, четвертым шагом в построении модели будет выделение свойств объектов.

В структуре наших представлений о свойстве легко выделить его основные характеристики:

- название (например, «температура по Цельсию»);
- ограничения на тип и диапазон значений (возможные значения температуры по Цельсию – вещественные числа от -273,15 до +∞);
- набор объектов, которые могут и/или должны являться носителем этого свойства (все физические тела обязательно имеют температуру, а период времени не может иметь температуры).

Связи объектов между собой тоже можно рассматривать как свойства – например, «владеет автомобилем». Эта идея достаточно естественна для нашего сознания. Ее следствием является разделение свойств на два типа. Одни служат для описания характеристик объектов, принимающих некие конкретные значения: числовые (температура, размер, длительность), строковые (название), логические (включен). Другие служат для отражения связей объектов между собой, выражая, например, информацию о том, что объект А является частью объекта Б, или что эти объекты взаимодействуют каким-то иным образом. Свойства первого вида мы назовем свойствами-*литералами*, второго типа – свойствами-связями.

В терминах логики свойства называются *предикатами*, и такое обозначение часто встречается в литературе о семантических технологиях. Для предикатов существует специальный формат записи. Так, запись ЯвляетсяДочерью (А, Б) говорит о том, что объект А связан с объектом Б отношением «является дочерью».

Принимая решение о представлении какой-либо характеристики объекта в виде свойства-литерала, следует проявлять большую осторожность. Например, в большинстве случаев будет неправильно создавать свойство «температура» для какого-либо материального объекта: значение температуры может быть

зафиксировано в акте измерения, и, следовательно, может быть свойством именно объекта класса «Измерение». Исключения возможны, но должны быть обоснованы. Похожее соображение относится и к свойствам-связям: при создании таких свойств каждый раз необходимо принимать решение, можно ли обойтись свойством, или необходимо создавать специальный объект для выражения связи. Вариант с объектом более универсален, но менее удобен при выполнении логических вычислений. Например, утверждение о том, что собака сторожит подозреваемого, может быть выражено как прямой связью между этими объектами, так и при помощи дополнительной сущности – такой, как экземпляр класса «Выполнение функции»: один объект (собака) выполняет определенную функцию (сторожит) по отношению к другому объекту (человек, выступающий в роли подозреваемого). Это позволит, например, определить интервал времени, в течение которого выполняется функция, а также сообщить информацию о том, что стало причиной такого поведения собаки, и к каким привело последствиям. Использование свойства-связи позволило бы только зафиксировать статическую картину.

В рамках рассматриваемой нами методологии моделирования определения и сами свойства могут находиться между собой в тех или иных отношениях, а также обладать характеристиками, позволяющими делать логические выводы. В этой главе мы просто перечислим их, а детально рассмотрим позднее:

- Эквивалентность (например, «длительность» и «продолжительность»).
- Инверсность. Это отношение актуально для двух свойств, выражающих отношения объектов. Например, отношение «является сыном» инверсно отношению «является родителем».
- Разъединенность. Два свойства-связи разъединены, если наличие одной из них исключает наличие другой. Например, свойства «является матерью» и «является дочерью» разъединены, так как А не может одновременно быть и матерью, и дочерью Б.

- Под-свойство. Так же, как и классы, свойства могут быть объединены в иерархии. Так, если свойство-связь «является дочерью» – под-свойство для «является родственником», то если А является дочерью Б, то верно и то, что А является родственником Б.
- Транзитивность. Если какое-либо свойство транзитивно, и известно, что объект А связан этим свойством с объектом Б, а Б – с В, то А связан тем же свойством с В. Например, свойство «является частью» – транзитивно. Значит, если двигатель является частью автомобиля, а поршень является частью двигателя, то верно и то, что поршень является частью автомобиля.
- Симметричность. Если А связан с Б симметричным свойством, это автоматически означает, что Б связан с А тем же свойством. Так, свойство «является супругом» симметрично, то есть если Иван является супругом Марии, то и Мария является супругой Ивана (для сравнения, свойство «является родителем» – не симметрично, т.к. обратное свойство имеет другой смысл).

Ограничения, налагаемые на значения, которые могут принимать свойства, можно формализовать так:

- Тип принимаемого значения. Для свойств-литералов характеризует тип литерала: число, строка, дата, логическое значение и др. Для свойств-связей перечисляет классы объектов, с которыми данная связь может связать некий объект.
- Диапазон возможных значений (числовые границы диапазона для свойств-литералов, перечисление определенного множества объектов для свойств-связей, и т.д.).
- Количество возможных значений. Это не интуитивное, но очень важное ограничение. Свойство «является супругом» может принимать только одно значение для граждан европейских стран, но несколько – для

арабских. Человек может иметь несколько имен, город – несколько названий. В то же время, свойство «температура» всегда будет иметь только одно значение, но обусловленное конкретной точкой измерения и моментом времени. Выражаясь точнее, в ряде случаев будет неправильно присвоить свойство «температура» физическим объектам; правильнее присвоить его объектам, относящимся к классу «акт измерения». Каждый такой объект будет иметь свойства, привязывающие измерение к моменту времени и точке, где оно выполнено. Свойство «температура» таких объектов сможет иметь только одно значение.

- Обязательность. Каждое свойство для каждого объекта может иметь 0, 1 или более обязательных значений. Если свойство имеет одно обязательное значение – это означает то, что как минимум одно значение свойства для каждого индивидуального объекта должно быть задано.

Задание ограничений в онтологической модели имеет серьезную специфику, связанную с тем, что основным способом их использования является задание принадлежности объекта к классам. Это весьма неочевидный момент, который мы подробно рассмотрим далее.

Ни перечисленные отношения и характеристики свойств, ни список возможных ограничений не являются исчерпывающими – они приведены, скорее, в качестве примеров логических утверждений, которые можно использовать при составлении моделей. В то же время, все эти виды утверждений имеют конкретное выражение при помощи технических средств семантических технологий, о чем мы поговорим в следующем разделе.

Следует подчеркнуть важное отличие используемого нами значения понятия «свойство» от свойств в объектно-ориентированном программировании. В ООП

свойство неразрывно связано с классом, в котором оно определено. В рассматриваемой нами парадигме моделирования свойства существуют независимо от классов; одно и то же свойство может относиться к объектам, принадлежащим различным классам. Более того, класс объекта может определяться в зависимости от того, какими свойствами он обладает. Кстати, понятие «класс» в ООП и в контексте нашего изложения имеет разный смысл: в ООП это шаблон, определяющий структуру информации об объектах определенного типа, и допустимые операции с ними. В нашем же случае класс – это всего лишь способ группировки объектов.

Тем не менее, для каждого свойства мы должны определить набор классов, к объектам которых применимо данное свойство, что было указано выше, при обсуждении основных характеристик свойства.

То, что одно свойство может относиться к объектам разных типов – контринтуитивная идея для программистов. Поле «длина» в таблице «Виды контейнеров» и «длина» в таблице «Товары» - это два разных поля, и для того, чтобы их сравнить между собой, необходима написанная программистом логика. В онтологической же модели и виды контейнеров, и товары имеют одно и то же свойство «длина». Это позволяет, например, найти одним запросом все предметы, обладающие определенной длиной, независимо от их типа.

В заключение напомним, что определения свойств, также как индивидуальные объекты и классы, должны иметь уникальные идентификаторы.

Кстати, принадлежность индивидуального объекта к какому-либо классу технически выражается при помощи свойства этого объекта. Утверждение о том, что некая сущность А *является экземпляром класса* Б выражается при помощи свойства (предиката) rdf:type, определенного в стандарте RDF.

Отнесение сущности к определенному стандартному виду – класс, индивидуальный объект, или свойство – также выражается при помощи свойства.

То есть, утверждение о том, что сущность А *имеет вид сущности модели Класс* выражается при помощи того же самого свойства (предиката) rdf:type.

Важно понимать, что для описания структуры и понятийного аппарата модели (определений классов и свойств) мы используем тот же самый инструментарий, что и для наполнения модели предметным содержанием. То есть, с технической точки зрения структура и содержание информации в семантических технологиях выражаются одними и теми же средствами, доступны при помощи одинаковых программных инструментов.

2.7. Пример онтологической модели

Итак, мы описали следующий процесс создания онтологической модели:

Рис. 10. Последовательность создания онтологической модели

Применим описанную методологию моделирования на практике. Вспомним рисунок 1, с которого мы начали обсуждение информационных моделей. Составим информационную модель изображенной на нем статической сцены, подразумевая, что она создается в целях отработки сценариев возможных действий при задержании подозреваемого.

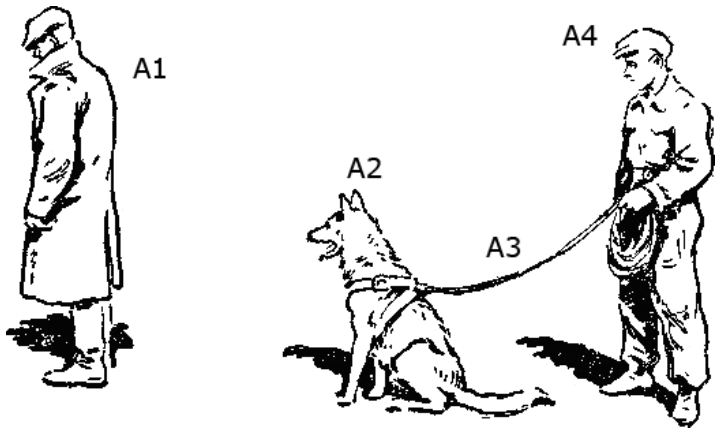

Рис. 11. Исходная информация для моделирования

1. Идентифицируем на этом рисунке четыре индивидуальных объекта: человек, собака, поводок, человек.
2. Присвоим им уникальные идентификаторы – соответственно, A1, A2, A3 и A4.
3. Определим набор классов, которые понадобятся для классификации выделенных нами объектов, и объединим их в иерархии:

По активности:	По типу:
Предмет	Человек
Действующее лицо	Подозреваемый
Статический предмет	Кинолог
	Собака
	Поводок

4. Зададим логические утверждения для созданных классов:

- Ни одно *действующее лицо* не является *предметом*, и наоборот.
- Классы *человек*, *собака*, *поводок* также имеют не пересекающиеся множества значений.
- Все *люди* (в пределах нашей модели) являются либо *подозреваемыми*, либо *кинологами*.
- Все *подозреваемые* являются *людьми*. Все *кинологи* являются *людьми*.

- Все *предметы* (в пределах нашей модели) являются либо *действующими лицами*, либо *статическими предметами*.
- Все *действующие лица* являются *предметами*. Все *статические предметы* являются *предметами*.

5. Классифицируем наши индивидуальные объекты:

- А1 относится к классам *Действующее лицо* и *Подозреваемый*.
- А2 относится к классам *Действующее лицо* и *Собака*.
- А3 относится к классам *Предмет* и *Поводок*.
- А4 относится к классам *Действующее лицо* и *Кинолог*.

6. Определим набор свойств-связей, которые понадобятся для описания взаимодействия объектов (названия свойств-связей выделены курсивом):

- А *держит в руке* Б.
- А *надет на* Б.
- А *сторожит* Б.
- А *находится под стражей* Б.

7. Опишем характеристики и ограничения для свойств.

- «*Держит в руке*» относится к объектам класса *Человек*, а его значениями являются объекты класса *Статический предмет*.
- «*Надет на*» относится к объектам класса *Действующее лицо*, а его значениями являются объекты класса *Предмет*.
- «*Сторожит*» и «*Находится под стражей*» относятся к объектам класса *Действующее лицо*, и их значениями являются также объекты класса *Действующее лицо*.

Все перечисленные свойства могут иметь от 0 до любого количества значений.

8. Опишем логические утверждения для свойств:

- Если А *сторожит* Б, то Б *находится под стражей* А.

9. Запишем значения свойств для наших объектов:

- A4 *держит в руке* A3. A3 *надет на* A2. A2 *сторожит* A1.

Таким образом, мы получили минимальное по содержанию, но полное по структуре описание сцены, изображенной на рис. 11. В приведенном виде модель позволяет нам сделать только один вывод – о том, что A1 *находится под стражей* A2. Для придания этой модели практического смысла ее нужно существенно расширить – например, описать те факты, что собака бросится на задержанного, если он попытается бежать, а кинолог даст команду и перестанет держать поводок. Полная с прагматической точки зрения модель должна содержать:

- описание целей (для подозреваемого цель – сбежать, для кинолога – предотвратить побег),
- возможных действий (бежать, напасть), и
- их последствий (обездвижен, ранен).

В такой модели цели, действия, последствия будут представлены при помощи классов и объектов. Таким образом, объектами в онтологических моделях могут выступать не только материальные предметы, но и действия, операции, состояния, ситуации. Например, объект «побег» будет относиться к классу «цели». Свойство «имеет цель» будет присуще каждому объекту класса «действующее лицо», и конкретно для объекта A1 его значение – «побег». Логические утверждения могут иметь такой, например, вид: «Если объект *обездвижен*, то он не может *бежать*»; «Если собака *держит* предмет, этот предмет *обездвижен*».

На такой модели можно будет провести имитацию развития событий при различных начальных условиях (например, при разных вариантах поведения подозреваемого), и получить в качестве результата инструкцию для кинолога по предотвращению побега.

Такая модель будет намного шире по содержанию, но будет состоять из сущностей тех же видов, которые мы использовали в нашем простейшем варианте модели.

3. Технологическое воплощение семантических моделей

3.1. Компьютерные технологии для семантического моделирования. RDF, RDFS и OWL

Семантические технологии – это набор способов представления и использования концептуализированной информации в электронном виде. Определение подчеркивает суть процесса использования семантических технологий: создание концептуального представления чего-либо, и его воплощение в электронной форме. Лучше всего понять особенности этих технологий можно в сравнении с другими способами представления информации.

Наиболее естественным и привычным для нас является *потоковое представление* (текст, изображения, видео, звуки). Представленная в потоковом виде информация – это сигнал, для расшифровки и воспроизведения которого используются специальные аппаратные и программные средства. Как правило, программное обеспечение не анализирует содержимое (смысл) такого сигнала, полностью возлагая задачу его интерпретации на пользователя. Несколько особняком стоит текстовое представление: текст может анализироваться при помощи статистических и иных методов, которые облегчают поиск и структурирование информации в нем. При этом осмысления текста не происходит, но используются некоторые инструменты, облегчающие работу с ним на концептуальном уровне для человека. Общим признаком, объединяющим все варианты потокового представления информации, является то, что воспринять ее смысл может только человек.

Для обработки структурированной информации используется *реляционное представление* (таблицы, базы данных). Суть этого подхода состоит в разделении информации на метаданные, описывающие тип и формат информационных

элементов, и собственно данные, несущие сведения о конкретных объектах, явлениях, их свойствах. Как правило, информация представляется в табличном виде. Каждая таблица хранит сведения о каком-либо типе объектов, явлений или связей. Строки таблицы представляют сведения о конкретных объектах или связях. Каждый столбец таблицы хранит информацию о каком-либо свойстве или конкретной связи объекта. Описания таблиц и столбцов являются метаданными, то есть сведениями о структуре данных, а содержимое таблиц – собственно данными. Наличие метаданных дает возможность строить программные алгоритмы обработки данных, то есть перекладывать часть задач по обработке информации на вычислительную машину.

Семантические технологии представляют собой следующий шаг в развитии машинно-читаемых представлений информации. Они дают возможность воплотить в электронном виде концептуальные модели, построенные по принципам, описанным в предыдущем разделе, позволяют передавать содержащуюся в этих моделях информацию и автоматически обрабатывать ее, в том числе – получать логические выводы на основании правил.

Технологическое воплощение семантических технологий несет на себе отпечаток истории их возникновения, о которой мы уже упоминали выше. Успешная реализация «семантической паутины» (Semantic Web) произвела бы революцию в механизмах поиска в Интернете, что имело бы очень серьезные экономические последствия для компаний, имеющих возможность влиять на его развитие. Возможно, это является одной из причин того, почему реальные успехи на этом поприще свелись к разработке нескольких микроформатов – крайне ограниченных онтологий, позволяющих публиковать на сайтах некоторую информацию в машинно-читаемом виде (адреса, сведения о товарах), и опять же крайне ограниченной поддержке этих микроформатов поисковыми машинами. Это можно сравнить с изобретением и разработкой микропроцессоров только для того,

чтобы создавать на них исключительно устройства, управляющие, например, движением лифтов. Конечно, необходимо отметить и достаточно высокий уровень технологической сложности создания семантических описаний всей публикуемой в интернете информации, проблемы выработки онтологического ядра и средств сопоставления элементов разных онтологий, без чего создание глобального графа потеряет смысл. Еще одним препятствием является изменение парадигмы работы с информацией как для ее авторов, так и для пользователей: если современные веб-сайты, по сути, мало чем отличаются по принципам предоставления информации от бумажных газет, то использование семантических технологий не имеет очевидного, привычного аналога в обычной жизни, что затрудняет их восприятие для пользователей.

Как бы то ни было, благодаря усилиям энтузиастов семантической паутины мы получили набор технологий, принятых в виде рекомендаций или стандартов W3C, и набор программного обеспечения с открытым исходным кодом, на котором можно решать задачи, далекие от довольно узких целей «семантической паутины». Потенциал этих технологий, на наш взгляд, серьезно недооценен. Тем интереснее разрабатывать методы их практического применения.

Технологии «семантической паутины» – далеко не первая реализация логических вычислений. Существует, например, язык Prolog, ряд других средств. Проблема состоит в их высокой сложности и малой пригодности для создания продуктов для конечного пользователя. У семантических технологий есть хороший шанс преодолеть этот разрыв.

Итак, рассмотрим состав «технологического стека» Semantic Web.

Прежде всего, необходим базисный способ выражения информации, представленной в онтологических моделях. Таким способом стал **триплет** – синтаксическая структура, состоящая из трех элементов:

Подлежащее Сказуемое Дополнение

Конечно, названия трем элементам триплета мы дали произвольно, но они отражают их суть. *Подлежащее* – это всегда какая-либо сущность модели, о которой сообщается информация. *Сказуемое* – свойство, значение которого мы хотим задать для этого объекта (в логике оно называется *предикатом*). Наконец, *дополнение* может быть или литералом (числовым, строковым значением), или другой сущностью. Разумеется, сущности и свойства задаются их уникальными идентификаторами. Выбранные нами названия частей триплета удобны еще и потому, что соответствуют грамматической структуре простейшей фразы-утверждения в естественном языке, например: «Собака – является – животным».

Вопрос для размышления

Запишите в виде триплетов сведения о том, что И.И. Иванов работает в компании ООО «Альфа», директором которой является П.П. Петров.

При помощи таких триплетов можно выразить *всю* информацию, содержащуюся в любой онтологической модели. Конечно, необходимо определить формализм, или набор правил, по которым составляются триплеты. Эти правила мы рассмотрим чуть позже, а пока продолжим обзор семантических технологий.

Одним из краеугольных камней онтологической модели являются **уникальные идентификаторы** объектов. В Semantic Web принято, чтобы эти идентификаторы имели вид URI – универсальных идентификаторов ресурсов, стандартизированных для сети Интернет. Таким образом, все идентификаторы в онтологических моделях получают вид:

http://имя-хоста/онтология#идентификатор

Вместо символа # для отделения названия онтологии от идентификатора конкретной сущности может использоваться слэш – /.

Нужно понимать, что такой способ записи идентификаторов – чистая условность. Идентификатор не имеет никакого отношения к протоколу http, «имя-хоста» может быть вымышленным и не соответствовать ни одному реально существующему сайту, названия онтологии и последняя (уникальная) часть идентификатора формируются произвольно, да и символа # перед идентификатором может не быть. В общем, URI вполне можно воспринимать просто как строку, которая формируется по определенному шаблону исключительно для обеспечения читаемости человеком.

Существуют общепринятые онтологии, такие как FOAF (friend of a friend), предназначенная для выражения информации о персонах. Такие онтологии имеют собственное пространство имен – первую часть идентификатора, такую как «http://имя-хоста/онтология». Например, для FOAF пространство имен – http://xmlns.com/foaf/0.1/. Собственные пространства имен используют и стандарты RDF, RDFS, OWL, с которыми мы познакомимся далее. В таких пространствах имен определяются идентификаторы сущностей, описанных в этих стандартах.

Следующий набор технологий, которые мы рассмотрим, применяется для сериализации онтологических моделей (т.е. для сохранения моделей в файлы). Здесь можно выделить логический и синтаксический уровень: логический уровень определяет принципы и правила передачи смыслового содержания онтологий, а синтаксический – способы преобразования этого смысла в последовательности символов.

Существуют несколько «языков» для записи семантических моделей, основными из которых являются RDF, RDFS и OWL. RDF и RDFS позволяют записывать простейшие факты об объектах, классах и свойствах. OWL описывает сложные взаимоотношения классов и свойств. Отношения между RDF/RDFS и OWL достаточно сложны: с одной стороны, OWL использует некоторые выражения предшествующих стандартов, такие как предикат принадлежности к типу (rdf:type); с другой стороны, он переопределяет такие их выражения, как тип сущности

«Класс» (owl:Class вместо rdfs:Class – здесь есть нюанс в определении класса для разных диалектов OWL). С третьей стороны, некоторые из таких переопределенных выражений используются реже, чем их аналоги из предшествующего стандарта: rdfs:subClassOf используется чаще, чем его аналог owl:subClassOf. В общем, на практике онтология обычно представляет собой смесь из выражений всех трех стандартов.

OWL имеет несколько «диалектов»: редко используемый, очень упрощенный OWL Lite, наиболее часто применяемый OWL DL (Description Logic), и богатый по выразительным возможностям OWL Full. Одно из различий между OWL DL и OWL Full состоит в том, что для OWL DL гарантируется вычислимость любого логического выражения, а для OWL Full – нет. С другой стороны, OWL Full позволяет, например, сделать какую-либо сущность классом и индивидуальным объектом одновременно, что бывает необходимо в сложных моделях. Далее нашем рассказе мы везде будем подразумевать использование OWL DL. Наконец, существует новая версия стандарта – OWL 2. В ней определены так называемые профили OWL, налагающие на возможности языка различные ограничения с целью оптимизировать скорость вычислений: OWL RL, OWL QL и др.

Стандартами предусмотрены разные синтаксисы, позволяющие сохранять OWL или RDF/RDFS в файл. Наиболее распространенным является хорошо известный XML. Используется также синтаксис Turtle, более лаконичный, и позволяющий легче «видеть» триплеты в коде.

Программное обеспечение для работы с RDF/RDFS/OWL включает редакторы, визуализаторы, а также машины логического вывода (английский термин – reasoner). Последние предназначены для того, чтобы проверять онтологии на наличие противоречий, а также автоматически делать выводы (т.е. продуцировать новые триплеты) на основании правил и имеющихся в модели фактов. Наиболее популярным свободно распространяемым редактором онтологий является Protégé, к которому в виде плагинов можно подключать средства визуализации, построения запросов к онтологии, получения выводов.

Понятно, однако, что работать с моделями в файловом режиме крайне неудобно, а в многопользовательском режиме – практически невозможно. Поэтому существуют программные продукты класса Triple store – хранилища триплетов, функционирующие аналогично базам данных. Наиболее известные решения такого класса – Apache Jena, Virtuoso, Sesame, GraphDB и множество других. В том числе, создать triple store можно на основе Oracle 11g. Хранилища триплетов имеют программные интерфейсы, позволяющие обращаться к ним из различных средств программирования, использовать машины логического вывода. Существует стандарт SPARQL, описывающий программный интерфейс и синтаксис запросов к онтологическим моделям; уже из его названия очевидна аналогия с языком SQL. Программная реализация SPARQL-интерфейса к содержимому хранилища триплетов называется точкой доступа SPARQL (SPARQL endpoint).

Работать со всем перечисленным ПО можно из любого языка программирования, но наиболее распространенной практикой является создание приложений на Java – благодаря наличию развитого программного интерфейса OWL API, реализуемого большинством семантического ПО.

Хранилища триплетов реализуют импорт онтологий из файлов RDF/RDFS/OWL, но обычно не поддерживают экспорт в них. Также нужно отметить, что не все продукты одинаково реализуют правила конвертации из OWL в «чистые» триплеты (см. документ «OWL 2 Web Ontology Language Mapping to RDF Graphs», http://www.w3.org/TR/2009/REC-owl2-mapping-to-rdf-20091027/), поэтому результаты импорта OWL в хранилище триплетов зависят от программной реализации.

Еще одним интересным классом семантического ПО являются системы, поддерживающие работу с контролируемым естественным языком. Они позволяют записывать выражения, однозначно транслирующиеся в OWL, на подмножестве естественного языка (английского, русского и т.д.). На естественном языке можно и задавать запросы машине логического вывода. Правда, синтаксические правила контролируемого языка в современных реализациях получаются достаточно

ограниченными. Примером такого продукта является FluentEditor от компании Cognitum.Существует и ПО, позволяющее восстанавливать онтологические модели из текста – например, ABBYY Compreno.

3.2. Простые онтологические модели: создание классов. Редактор Protégé

Обзор возможностей OWL мы приведем далее. Пока же нашей задачей является формирование целостного представления о взаимосвязи семантических технологий, и о воплощении идей семантического моделирования при помощи существующих программных реализаций. Поэтому мы будем рассматривать построение и использование онтологических моделей на примерах, включающих реализацию моделей сразу в нескольких программных средах.

Для начала нужно разобраться с тем, как представляются в онтологических моделях сущности, описанные нами в первом разделе: индивидуальные объекты, классы, определения и значения свойств. Начнем мы с определения класса. Факт существования класса констатируется одним триплетом:

Подлежащее	**Сказуемое (предикат)**	**Дополнение**
Уникальный идентификатор класса	Имеет тип	Класс

В следующих таблицах с триплетами мы не будем приводить заголовок «подлежащее – сказуемое – дополнение», потому что структура триплета всегда одинакова.

Словами «Имеет тип» мы обозначили предикат, определенный в стандарте RDF, который используется для указания типа сущностей онтологических моделей. URI этого свойства выглядит так: http://www.w3.org/1999/02/22-rdf-syntax-ns#type, или сокращенно – rdf:type.

Словом «Класс» мы заменили URI определения «класс», заданного в стандарте OWL. Оно имеет такой вид: http://www.w3.org/2002/07/owl#Class, или сокращенно – owl:Class.

Заметим, что если бы мы оставались в рамках стандарта RDF/RDFS, то определение класса выглядело бы так: http://www.w3.org/2000/01/rdf-schema#Class (rdfs:Class).

Уникальный идентификатор конкретного класса мы можем придумать сами. Пусть он будет таким: http://example.ru/sample#OurFirstClass. Тогда полностью наш триплет будет выглядеть так:

http://example.ru/sample#OurFirstClass	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class

Для читаемости подобных записей они сокращаются путем использования префиксов, заменяющих «общую» часть URI. Для терминов RDF, RDFS и OWL есть соответствующие стандартные префиксы, а для нашей онтологии мы можем определить свой. Например, объявим префикс my_ontology, который будет заменять часть URI http://example.ru/sample#. Тогда приведенная выше запись станет такой:

my_ontology:OurFirstClass	rdf:type	owl:Class

Читается куда лучше, не так ли? Суть этого триплета, повторим, состоит в констатации того факта, что сущность http://example.ru/sample#OurFirstClass является классом. Заметим еще раз, что приведенная нотация является лишь одним из способов выразить эту информацию.

В начале RDF-файла в XML-синтаксисе префиксы могут быть объявлены так:

```
<rdf:RDF xmlns="http://example.ru/sample#"
     xml:base="http://example.ru/sample"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
```

```
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
```

Посмотрим, как создать это утверждение в редакторе, поддерживающем стандарт OWL. Для первых упражнений мы будем использовать Protégé – свободно распространяемый редактор, разработанный Стэнфордским университетом.

Открыв редактор, перейдем на закладку Classes, нажмем кнопку Add Subclass в левом списке, и введем название – OurFirstClass. Заметим, что редактор сделал наш класс подклассом предустановленного класса Thing, к которому относится все сущее. Впрочем, этот факт никак не отразится в файлах, которые мы получим после сохранения модели.

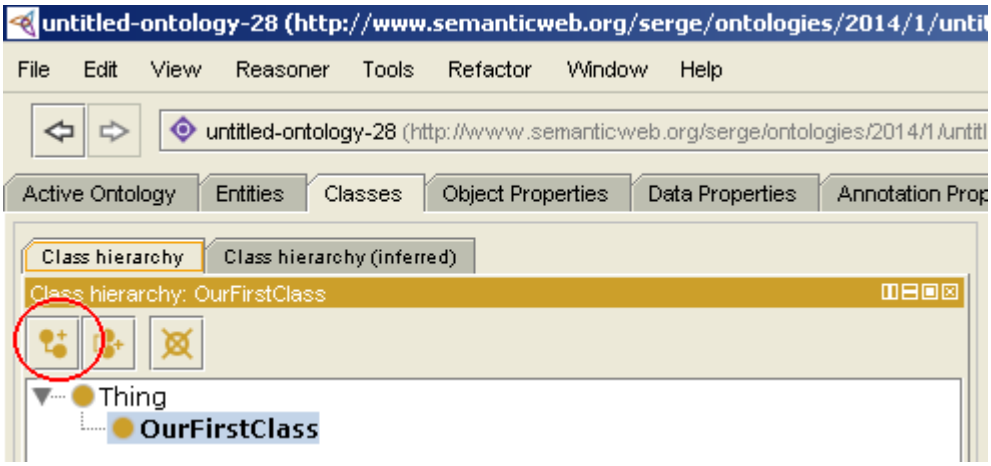

Рис. 12. Создание класса в редакторе Protégé

Сохраним нашу модель, чтобы посмотреть, как она будет выглядеть в файле. Обратим внимание, что редактор предлагает нам выбрать из набора форматов, среди которых есть RDF/XML и OWL/XML. Если мы выберем RDF/XML, то файл примет такой вид (заголовки не приведены):

```
<owl:Class rdf:about="http://example.ru/sample#OurFirstClass"/>
```

Если выбрать OWL/XML, то содержательная часть файла будет выглядеть так:

```
<Declaration>
    <Class IRI="#OurFirstClass"/>
</Declaration>
```

Таким образом, одна и та же модель может быть сериализована в файл XML совершенно по-разному, в зависимости от конкретного стандарта.

Интересно также пронаблюдать за тем, что произойдет при импорте полученных файлов в хранилище триплетов через SPARQL-интерфейс. Для этого воспользуемся продуктом Apache Jena, точнее – входящей в состав этого фреймворка SPARQL точкой доступа, которая называется Fuseki. Fuseki, как и другие точки доступа, имеет собственный веб-интерфейс для выполнения запросов, который может быть использован как человеком, так и программным продуктом. Войдя в этот интерфейс (если установить Fuseki на локальный компьютер, то ссылка для входа в него будет – http://localhost:3030), нажмем на ссылку Control panel, и выберем набор данных (dataset). При работе с первой версией Fuseki мы окажемся на странице, содержащей три формы, соответствующей службам SPARQL:

- SPARQL Query – запросы на извлечение информации,
- SPARQL Update – запросы на изменение информации в хранилище,
- File upload – загрузка файла RDF или OWL.

При работе со второй версией мы увидим пункты меню Dataset, за которым скрываются возможности выполнения запросов и загрузки файлов, и Manage datasets, через который можно создавать новые наборы данных и управлять ими.

Воспользуемся сначала формой загрузки файла для того, чтобы поместить в точку доступа содержимое нашего файла, экспортированного из Protégé в формате RDF/XML. Выберем файл и нажмем кнопку Upload. В качестве результата Fuseki покажет нам количество созданных триплетов – их будет два.

Перейдем на страницу запроса и выполним простейший SPARQL-запрос, который вернет все содержимое модели (синтаксис SPARQL-запросов мы обсудим далее):

```
SELECT * WHERE { ?a ?b ?c }
```

Мы увидим, что результатом будет тот самый триплет, который мы приводили выше:

http://example.ru/sample#OurFirstClass	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class

Вернемся к рассказу об основных приемах создания моделей. Первое действие, которое нам захочется выполнить с классами – определение подклассов. В Protégé для этого нужно установить курсор на класс-родитель и нажать кнопку Add subclass. В RDF/XML факт наличия класса-родителя будет отражен таким образом:

```
<owl:Class rdf:about="http://example.ru/sample#SecondClass" />
      <rdfs:subClassOf="http://example.ru/sample#OurFirstClass" />
</owl:Class>
```

При импорте такого определения в Triple store образуются два триплета:

http://example.ru/sample#SecondClass	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#Class
http://example.ru/sample#SecondClass	http://www.w3.org/2000/01/rdf-schema#subClassOf	http://example.ru/sample#OurFirstClass

Обращаться к классам по их URI не очень-то удобно, даже если определить префикс. Нужно учитывать, что URI, например, не может содержать пробелов и ряд

других спецсимволов – значит, все «имя» класса должно представлять собой одно слово. Если мы хотим, чтобы с нашими классами работали пользователи, не знакомые с семантическими технологиями, желательно спрятать от них тот факт, что уникальным идентификатором класса является URI, а не просто строка. Для этого нужна возможность задавать для классов читаемые имена, предпочтительно – на нашем родном языке.

В Protégé для этого предназначена закладка Annotations в свойствах класса. Нажав на расположенную в ней кнопку «+», увидим диалоговое окно создания описаний:

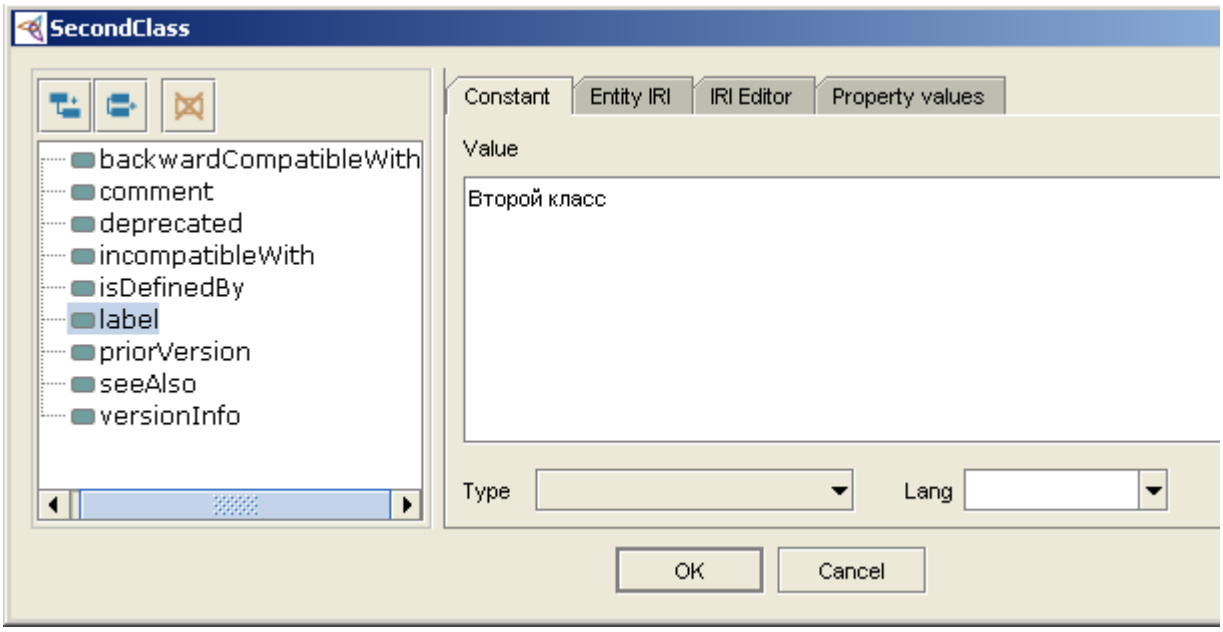

Рис. 13. Описание класса в редакторе Protégé

Слева перечислены несколько стандартных свойств, используемых для описания элементов модели. Они являются встроенными определениями, предусмотренными стандартами RDF/OWL. Важнейшим из них является label (ярлык или метка) – это свойство как раз и предназначено

Остальные виды аннотаций пригодятся для организации совместной работы с онтологией, поддержки ее версионности, отслеживания происхождения каждого элемента онтологии.

для задания читаемых имен элементов. Часто используется также comment – комментарий, расширенное описание элемента.

Выбрав свойство label, в поле ввода в правой части окна вводим его значение. Обратите внимание, что внизу предусмотрена также возможность задать тип значения (в данном случае это будет строка, string) и/или язык (русский). Тип значения задавать не обязательно, а вот язык имеет смысл выбрать в том случае, если необходимо задать названия элемента на нескольких языках. Кстати, эта особенность RDF/OWL делает его очень удобным для реализации многоязычных приложений. В XML/RDF наше описание будет выглядеть так:

```
<owl:Class rdf:about="http://example.ru/sample#SecondClass" />
      <rdfs:label>Второй класс</rdfs:label>
      <rdfs:subClassOf="http://example.ru/sample#OurFirstClass" />
</owl:Class>
```

Триплет, соответствующий описанию, выглядит так:

http://example.ru/sample#SecondClass	http://www.w3.org/2000/01/rdf-schema#label	"Второй класс"

Имена (label) и прочие аннотации присваиваются одинаковым образом всем основным элементам семантической модели – классам, индивидуальным объектам и свойствам, значениям свойств, логическим аксиомам, ограничениям и др.

3.3. Простые онтологические модели: индивидуальные объекты и свойства

Как мы помним, основными элементами семантических моделей являются классы, индивидуальные объекты и свойства. Рассмотрим создание индивидуальных объектов (иначе называемых, хоть и не совсем корректно,

экземплярами), определений свойств, и присвоение значений свойствам определенных экземпляров.

В Protégé сущности всех типов создаются похожим образом, только для этого используются разные закладки редактора:

- Classes – классы,
- Individuals – индивидуальные объекты,
- Data properties – свойства-литералы,
- Object properties – свойства-указатели на объекты.

Закладка Entities позволяет увидеть все эти элементы одновременно.

В RDF/XML полученные сущности будут отличаться типом, соответственно:

- http://www.w3.org/2002/07/owl#Class
- http://www.w3.org/2002/07/owl#NamedIndividual
- http://www.w3.org/2002/07/owl#DatatypeProperty
- http://www.w3.org/2002/07/owl#ObjectProperty

Например, триплет с объявлением индивидуального объекта может выглядеть так:

`http://example.ru/sample#SecondClassObject`	`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`	`http://www.w3.org/2002/07/owl#NamedIndividual`

Конечно, при создании индивидуального объекта обычно указывают, к каким классам он относится (мы помним, что объект может входить в несколько классов одновременно). В редакторе Protégé для этого нужно выбрать

Включив объект в SecondClass, мы тем самым автоматически включили его и в надкласс OurFirstClass. Но чтобы получить эту информацию в явном виде, нужно подключить машину логического вывода, о чем мы расскажем далее.

созданный объект на вкладке Individuals, и нажать кнопку «+» напротив слова Types (правое нижнее окно). Откроется диалоговое окно выбора класса. Результат выполнения этой операции выглядит так:

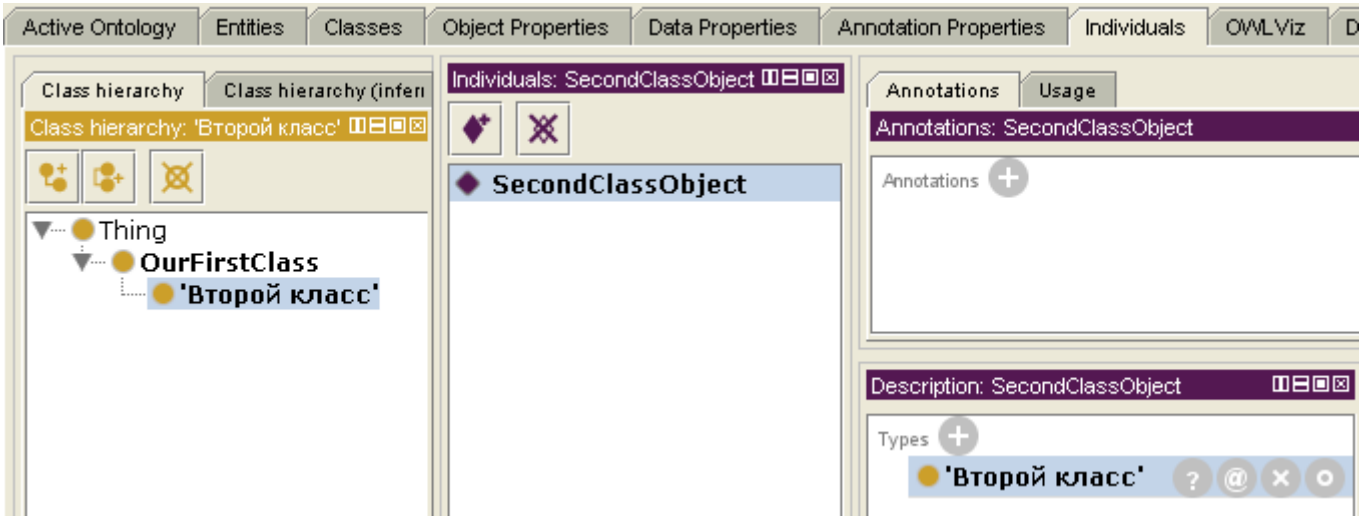

Рис. 14. Индивидуальный объект – член класса в редакторе Protégé

Кстати, обратите внимание, что для «Второго класса» Protégé показывает в своем интерфейсе значение его свойства label, а не уникальную часть идентификатора URI.

В RDF/XML факт вхождения объекта в класс отразится следующим образом:

```
<owl:NamedIndividual rdf:about="http://example.ru/sample#SecondClassObject">
    <rdf:type rdf:resource="http://example.ru/sample#SecondClass"/>
</owl:NamedIndividual>
```

Триплеты в этом случае будут выглядеть так:

http://example.ru/sample#SecondClassObject	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://www.w3.org/2002/07/owl#NamedIndividual
http://example.ru/sample#SecondClassObject	http://www.w3.org/1999/02/22-rdf-syntax-ns#type	http://example.ru/sample#SecondClass

Обратите внимание на то, что определение свойства во втором столбце – одинаково в обоих триплетах! То есть тот факт, что элемент модели SecondClassObject относится к предопределенному стандартом классу NamedIndividual, и является членом определенного в данной онтологии класса SecondClass, выражается при помощи одного и того же предопределенного стандартом свойства. Из этого следует, кстати, что код RDF/XML можно переписать так:

```
<SecondClass rdf:about="http://example.ru/sample#SecondClassObject">
    <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#NamedIndividual"/>
</SecondClass>
```

Результат импорта этого кода в triple store будет точно таким же, как в предыдущем случае.

Перейдем к рассказу о том, как в модели задаются свойства и их значения. Определение свойства, значением для которого является литерал, происходит так. В редакторе Protégé нужно перейти на закладку Data Properties, создать новое свойство – кстати, обратите внимание, что у свойств может быть своя иерархия. После этого, используя правое нижнее окно, можно задать два важных атрибута свойства: Domain и Range. Domain – это класс, к экземплярам которого применимо данное свойство. Range – это тип значений, которые свойство может принимать. Соответственно, значение для Domain выбирается из списка классов, существующей в нашей модели, а Range – из определенных стандартом типов данных (используются типы данных XSD). Обратим внимание на то, что семантическое ПО интерпретирует эти значения не совсем очевидным образом. Например, задание Domain не является ограничением, заставляющим присваивать значения этого свойства только объектам указанного класса. Вместо этого reasoner будет считать, что любой объект, которому присвоено какое-либо значение этого свойства, является экземпляром указанного класса!

И классов-носителей, и диапазонов значений у свойства может быть несколько, но здесь нас подстерегает ее один неочевидный момент: в соответствии со стандартом RDFS, если заданы несколько классов-носителей, то значением свойства смогут обладать только те индивидуальные объекты, которые относятся сразу ко всем перечисленным классам, т.е. находятся на их пересечении. Использовать объединение классов в качестве диапазона можно только путем создания специального класса-объединения с помощью выражения owl:unionOf, о чем мы расскажем ниже.

В окне редактора результат создания свойства выглядит так:

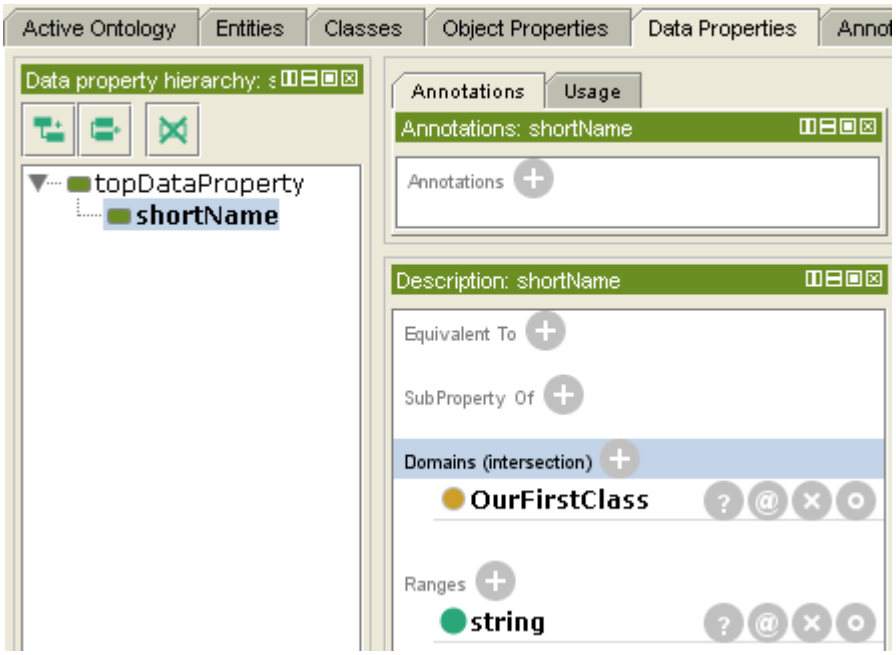

Рис. 15. Свойство-литерал в редакторе Protégé

Аналогичным образом создаются свойства-указатели на объекты. Для этого используется закладка Object Properties. Диапазон значений в этом случае также выбирается из числа классов, определенных в онтологии.

Для свойств обоих типов можно также задать ограничения на число значений, которые может иметь данное свойство для конкретного объекта (например, имен у

человека может быть несколько, а дата рождения – только одна). Для этого используется понятие Cardinality. Чтобы создать ограничение, нужно нажать на кнопку «+» напротив слова Ranges, и перейти на закладку Object (или Data) Restriction creator в появившемся диалоговом окне. Здесь нужно выбрать свойство, на которое накладывается ограничение, затем класс(ы) или тип(ы) принимаемых значений; внизу окна выбирается тип ограничения – минимальное или максимальное количество значений, и вводится число. Например, если установить min cardinality = 1, это будет означать, что свойство должно иметь по меньшей мере одно значение для каждого объекта. По умолчанию любое свойство может иметь любое количество значений для каждого объекта.

На самом деле, с ограничениями все не так просто – установленное ограничение не помешает присвоить свойствам любые значения. Мы вернемся к этому вопросу в главе 6.

Указанные ограничения используются машиной логического вывода прежде всего таким же образом, как Domain и Range, то есть для определения того, к какому классу относится объект-обладатель свойства. Но если создать в онтологии явное противоречие – например, задать принадлежность индивидуального объекта A к классу B, для свойства C задать Domain = B, минимальное число значений max cardinality = 1, а затем указать два значения свойства C для объекта A – reasoner выдаст предупреждение об ошибке, неконсистентности онтологии. Любопытно, что сделать так же с min cardinality не получится – если задать min cardinality у свойства C = 1, и не указать ни одного значения свойства C для объекта A, ошибки не произойдет. В OWL действует так называемая «парадигма открытого мира», с которой мы еще не раз столкнемся на страницах этой книги. Она состоит в предположении, что в онтологии задана не вся известная информация, а только ее часть. Таким образом, reasoner предполагает, что объект A обладает значением свойства C, но этот факт просто не отражен в онтологии.

Кстати, более простой и надежный способ указать, что у свойства должно быть только одно значение для каждого обладающего им объекта, состоит во включении определения свойства в предопределенный стандартом класс

FunctionalProperty (в Protégé это делается путем установки галочки Functional в определении свойства).

Результат сохранения определения свойства-литерала в формат RDF/XML выглядит так:

```
<owl:DatatypeProperty rdf:about="#shortName">
    <rdfs:domain rdf:resource="#OurFirstClass"/>
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>
```

Для свойства-указателя на объект:

```
<owl:ObjectProperty rdf:about="#ObjectProp">
    <rdfs:domain rdf:resource="#OurFirstClass"/>
    <rdfs:range rdf:resource="#SecondClass"/>
</owl:ObjectProperty>
```

Конечно, для читаемости определениям свойств следует присваивать label.

Информацию об ограничениях на число значений Protégé сохраняет в RDF/XML не самым очевидным образом, с использованием анонимных элементов онтологии – так называемых «пустых узлов», с которыми мы познакомимся далее:

```
<owl:ObjectProperty rdf:about="#ObjectProp">
    <rdfs:domain rdf:resource="#OurFirstClass"/>
    <rdfs:range rdf:resource="#SecondClass"/>
    <rdfs:range>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#ObjectProp"/>
            <owl:onClass rdf:resource="#SecondClass"/>
            <owl:minQualifiedCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minQualifiedCardinality>
        </owl:Restriction>
    </rdfs:range>
</owl:ObjectProperty>
```

Нам осталось научиться присваивать значения свойств индивидуальным объектам. В редакторе Protégé для этого предназначена область Property Assertions на странице просмотра индивидуального объекта. Нажатием кнопки «+» напротив Data property assertion и Object property assertion мы можем вызвать диалоговые окна, в которых нужно выбрать свойство, и ввести для него значение. Результат выполнения этой операции выглядит так:

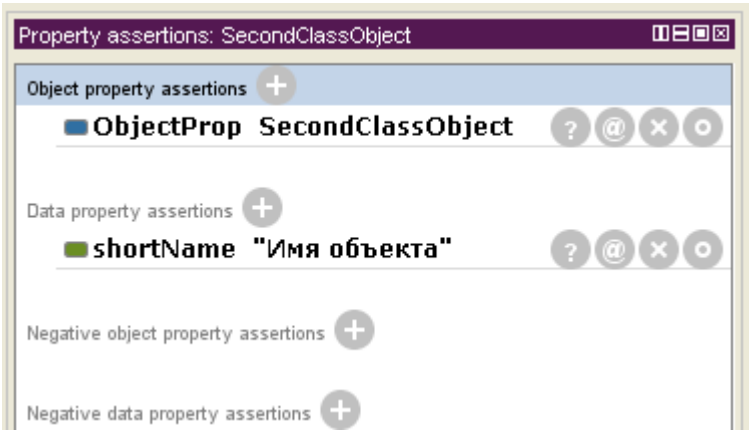

Рис. 16. Присвоение свойств индивидуальным объектам

В формате RDF/XML эта информация будет выглядеть так:

```
<owl:NamedIndividual rdf:about="#SecondClassObject">
    <rdf:type rdf:resource="#SecondClass"/>
    <shortName>Имя объекта</shortName>
    <ObjectProp rdf:resource="#SecondClassObject"/>
</owl:NamedIndividual>
```

Для свойств-литералов можно (но не обязательно) задать тип значения, а для строк – еще и язык. Например, если мы укажем явным образом, что значение свойства shortName является строкой – в файле это отразится так:

```
<shortName rdf:datatype="&xsd;string">Имя объекта</shortName>
```

В триплетах, с учетом указания типа, значения свойств shortName и objectProp отразятся так:

http://example.ru/sample#SecondClassObject	http://example.ru/sample#shortName	"Имя объекта" ^^<http://www.w3.org/2001/XMLSchema#string>
http://example.ru/sample#SecondClassObject	http://example.ru/sample#ObjectProp	http://example.ru/sample#SecondClassObject

Обратите внимание на то, как преобразуется информация о типе значения при импорте в триплеты: суффикс ^^<http://www.w3.org/2001/XMLSchema#string> добавляется к самому значению.

Следует отметить несколько неприятных, на первый взгляд, особенностей программного обеспечения, работающего с семантическими моделями. Так, Protégé при выключенной машине логического вывода (reasoner) – а по умолчанию она выключена – не производит никакой логической обработки содержимого модели, поэтому формализмы RDF и OWL, включая ограничения, не имеют для них ровным счетом никакого значения. Хранилища триплетов по умолчанию работают так же. Triple store спокойно примет ошибочные и абсурдные определения любых моделей. Вместо URI можно передавать произвольные строки – это является формальным нарушением стандарта, однако triple store совершенно не беспокоит. Для него триплет – это тройственная структура, любой элемент которой может принимать любые значения. Справляются с описанной «проблемой» (или, точнее, не вполне очевидным, но стандартным поведением ПО) машины логического вывода, с которыми мы познакомимся далее.

3.4. Простые онтологические модели: общая картина

Мы рассмотрели способы создания основных элементов семантических моделей в простейшем редакторе – Protégé; одновременно мы познакомились с тем, как такие модели сериализуются в формат RDF/XML, и что происходит с ними при

загрузке в хранилище триплетов. Целостность понимания этих механизмов очень важна для успешной работы с программным воплощением семантических моделей.

Чтобы привести эти знания в общую систему, а заодно получить немного новой информации, окинем взглядом взаимосвязь изученных нами элементов семантических моделей. Рассмотрим пример простой онтологии. Пусть мы хотим выразить в модели информацию о том, что объекты класса Customer (клиент) обладают свойством Name (название), и свойством Child, которое устанавливает отношения дочерняя-родительская организация между клиентами. Имеются две компании – ЗАО «Альфа» и ООО «Бета», причем «Бета» является дочерней структурой «Альфы». Всю эту модель можно представить так:

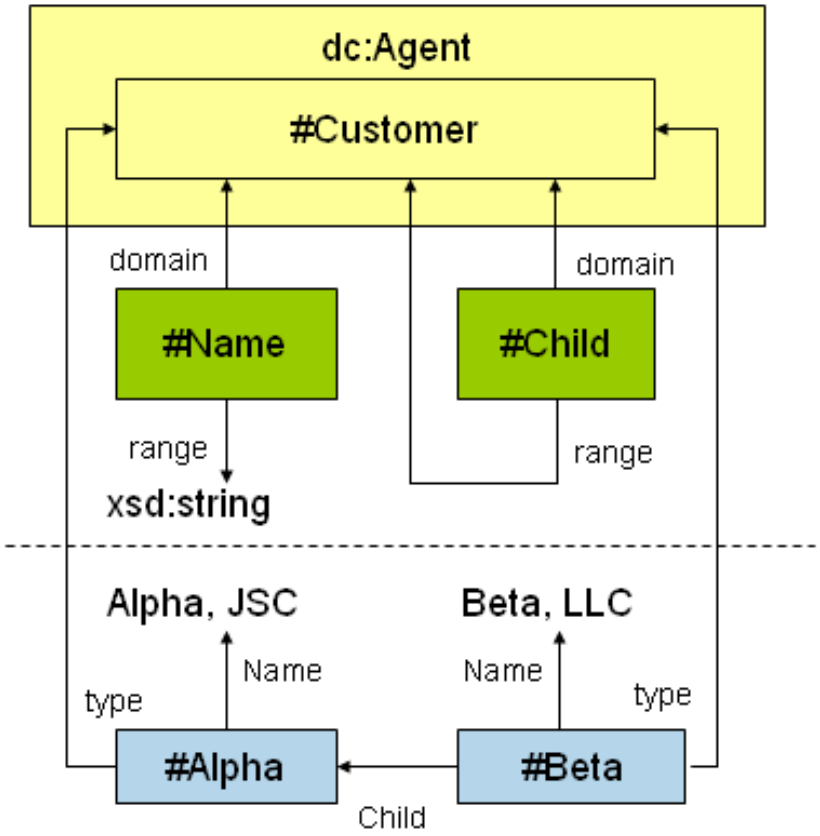

Рис. 17. Пример целостной семантической модели

Префиксы на этой диаграмме не показаны в целях читаемости.

Обратим внимание на пунктирную линию: то, что находится выше нее, представляет собой определения классов и свойств (TBox), а то, что ниже – информацию об индивидуальных объектах и их свойствах (ABox).

> *Существует немало стандартных онтологий, но в большинстве случаев оказывается удобнее создать полностью собственную модель, чем заимствовать определения из сторонних. Исключение – ситуация, когда хранящейся в нашей модели информацией надо будет обмениваться с другими системами.*

В верхней части диаграммы мы видим определение класса Customer (клиент). Он является подклассом dc:Agent. Префикс dc соответствует стандартной онтологии Dublin Core. Эта онтология представляет собой интересную попытку дать общеупотребимые определения для сущностей, встречающихся практически в любой семантической модели – таких, как «действующее лицо», «персона» и др. Делая наш класс Customer подклассом определенного в стандартной онтологии класса Agent, мы сообщаем потенциальным внешним потребителям нашей модели информацию о том, как следует воспринимать этот класс.

Ниже показаны два свойства – #Name и #Child. Они представляют собой отдельные элементы модели. Как мы помним из предыдущей главы, свойства имеют две важные характеристики – domain, указывающий, экземплярам каких классов присуще это свойство (в данном случае domain обоих свойств указывает на класс Customer), и range, определяющий тип принимаемого значения. Name – строковое свойство, а значением свойства Child являются также объекты класса Customer.

Ниже пунктирной линии на рисунке находятся определения двух индивидуальных объектов – #Alpha и #Beta. Предопределенное стандартом свойство type указывает на класс Customer. У каждого объекта имеется имя, представленной строковым литералом. Наконец, объект Beta имеет свойство Child, которое указывает на Alpha, поскольку Beta является дочерней организацией компании Alpha. Посмотрим на получившиеся триплеты:

<http://example.com/#Customer>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#Class>
<http://example.com/#Customer>	<http://www.w3.org/2000/01/rdf-schema#subClassOf>	<http://purl.org/dc/terms/Agent>
<http://example.com/#Name>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#DatatypeProperty>
<http://example.com/#Name>	<http://www.w3.org/2000/01/rdf-schema#range>	<http://www.w3.org/2001/XMLSchema#string>
<http://example.com/#Name>	<http://www.w3.org/2000/01/rdf-schema#domain>	<http://example.com/#Customer>
<http://example.com/#Child>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#ObjectProperty>
<http://example.com/#Child>	<http://www.w3.org/2000/01/rdf-schema#range>	<http://example.com/#Customer>
<http://example.com/#Child>	<http://www.w3.org/2000/01/rdf-schema#domain>	<http://example.com/#Customer>
<http://example.com/#beta>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#NamedIndividual>
<http://example.com/#beta>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://example.com/#Customer>
<http://example.com/#beta>	<http://example.com/#Child>	<http://example.com/#alpha>
<http://example.com/#beta>	<http://example.com/#Name>	"Beta, LLC"
<http://example.com/#alpha>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#NamedIndividual>
<http://example.com/#alpha>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://example.com/#Customer>
<http://example.com/#alpha>	<http://example.com/#Name>	"Alpha, JSC"

Верхняя часть таблицы, выделенная светло-серым фоном, соответствует верхней части диаграммы, показанной над пунктирной линией. Эту часть модели, TBox, можно условно назвать «справочными данными» – она описывает структуру информации. Нижняя часть, ABox соответствует наполнению модели, «фактографическим данным». Такое деление очень условно, поскольку часто то, что является фактографическими данными в одной модели, становится

справочными в другой; технологически никакой разницы между этими типами данных нет. Тем не менее, с точки зрения восприятия и понимания модели, такое разделение может оказаться полезным.

4. Технологии использования онтологических моделей в информационных системах

4.1. Онтологическая модель как граф. Язык SPARQL

Граф – это математический термин, означающий набор *вершин*, и соединяющих их *ребер*. Простейший граф выглядит так:

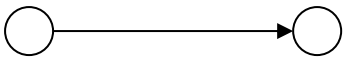

Рис. 18. Простейший граф

Обратите внимание, что ребро является направленным.

Очевидно, что структура такого графа идеально соответствует структуре триплета: вершины – это подлежащее и дополнение, а ребро – сказуемое. Например, триплет «Собака является животным» можно изобразить так:

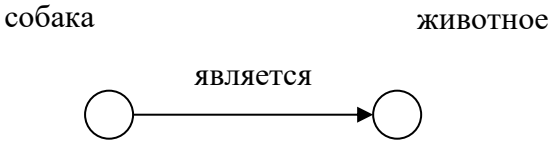

Рис. 19. Логическое выражение, представленное в виде графа

Таким образом, любую онтологию можно изобразить в виде графа. Проиллюстрируем это на примере из предыдущей главы.

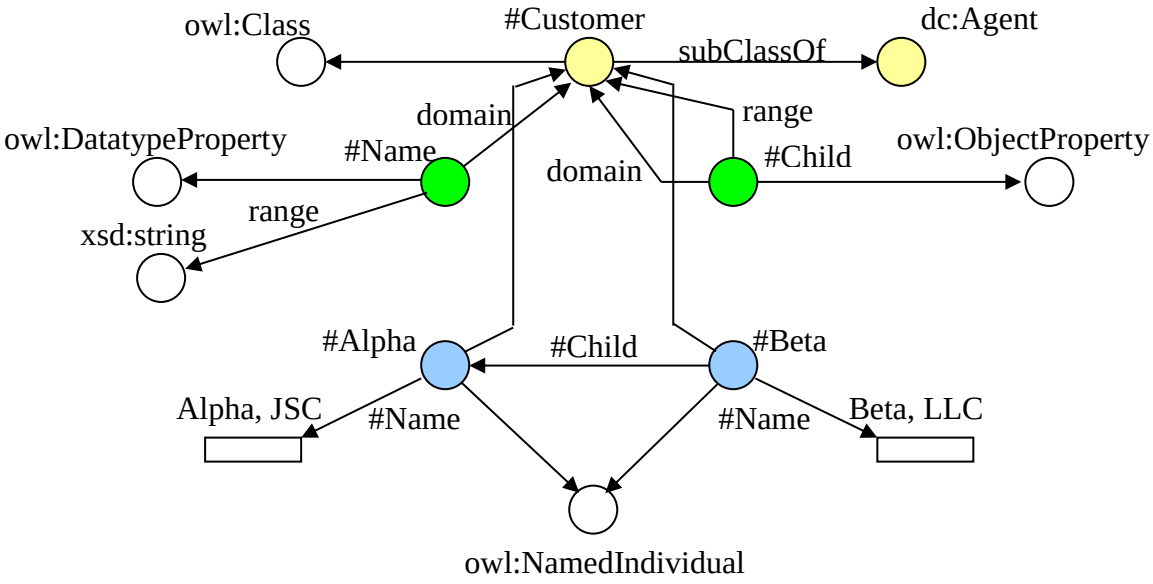

Рис. 20. Модель, представленная в виде графа

Глядя на подобные изображения, важно понимать, что вершина соответствует, фактически, уникальному идентификатору объекта (за исключением вершин-литералов, которые мы показали прямоугольниками). Она не содержит никакой другой информации; все приписанные ей свойства задаются отдельными ребрами графа. Некоторые вершины мы выделили цветом только для удобства восприятия.

Правильность изображения нашей модели легко проверить, посчитав число ребер. Их 15, столько же, сколько триплетов в таблице из предыдущей главы.

Поскольку полное графическое представление модели обычно оказывается слишком сложным, его упрощают – не показывают ребра, отображающие тип элемента, указывающие на литералы и т.д.

Заметим также, что в нашей модели сущности-свойства (#Name, #Child) присутствуют в двух ролях – в качестве вершин, что соответствует их описанию, и в качестве ребер, когда выполняется присвоение значения свойства конкретному объекту.

Такое представление семантической модели определяет способ построения запросов к ней, реализованный в языке SPARQL. Мы уже видели пример простейшего SPARQL-запроса:

```
SELECT * WHERE { ?a ?b ?c }
```

Слова SELECT * WHERE понятны всем, кто знаком с SQL – это выборка всех результатов из набора строк (в SQL) или триплетов (в SPARQL), отвечающих условиям отбора. Условия отбора, в случае SPARQL – это выражение, заключенное в фигурные скобки. Оно определяет *паттерн*, или шаблон, которому должны соответствовать триплеты. Значения, начинающиеся на вопросительный знак – это переменные. Они могут принимать любые значения, соответствующие условиям запроса; эти значения и будут возвращены в результате запроса. Поскольку все три позиции в нашем запросе заняты переменными – все три элемента триплета могут быть любыми. Значит, такой запрос вернет все содержимое онтологии (не следует злоупотреблять такими запросами на больших онтологиях).

Приведем несколько примеров осмысленных SPARQL-запросов. Предположим, мы хотим узнать, какой тип имеет объект #Alpha:

```
SELECT * WHERE { <http://example.com/#alpha> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?c }
```

Для читаемости перед запросом можно объявить префиксы. Тогда наш запрос станет выглядеть так:

```
PREFIX   rdfs: <http://www.w3.org/2000/01/rdf-schema>

SELECT * WHERE { <http://example.com/#alpha> rdfs:type ?c }
```

Результатом запроса будут значения переменной ?c – единственной оставшейся неизвестной:

?c
<http://www.w3.org/2002/07/owl#NamedIndividual>
<http://example.com/#Customer>

Чтобы получить список всех членов класса Customer, нужно выполнить такой запрос:

```
SELECT * WHERE { ?cust rdfs:type <http://example.com/#Customer> }
```

Результатом будут URI клиентов:

?cust
<http://example.com/#alpha>
<http://example.com/#beta>

Предположим, что мы хотим одновременно с уникальным идентификатором клиентов получить их имена. Тогда в запросе нужно будет указать паттерн, соответствующий двум ребрам графа:

```
SELECT * WHERE {  ?cust   rdfs:type            <http://example.com/#Customer>.
                  ?cust   <http://example.com/#Name>  ?name }
```

Как мы видим, паттерны триплетов отделяются друг от друга точкой. Результат выполнения этого запроса будет таким:

?cust	?name
<http://example.com/#alpha>	“Alpha, JSC”
<http://example.com/#beta>	“Beta, LLC”

Еще более усложним запрос. Пусть нам нужны сразу имена вышестоящих структур для каждой компании. При этом URI мы не хотим видеть в результатах

запроса. Нужно принимать во внимание, что у некоторых компаний может не быть вышестоящих структур, так что у паттерна появляется опциональная часть, которая может присутствовать не у всех результатов. Запрос будет выглядеть так:

```
SELECT ?name ?parent_name WHERE {
                        ?cust rdfs:type <http://example.com/#Customer>.
                        ?cust <http://example.com/#Name> ?name.
            OPTIONAL {  ?cust <http://example.com/#Child> ?parent.
                        ?parent <http://example.com/#Name> ?parent_name }
                    }
```

Обратим внимание на следующие детали синтаксиса:

- Вместо * в списке выбираемых значений можно указать через пробел имена тех переменных, которые нужны в списке возвращаемых значений. Остальные переменные используются в структуре паттернов запроса, но выданы в виде результатов не будут.
- Клауза OPTIONAL (в запросе их может быть несколько) используется для задания дополнительных паттернов, которым могут, но не обязаны соответствовать результаты запроса. Это очень полезно в случаях, когда выбираются объекты вместе с их свойствами, но свойства заданы не у каждого объекта. Если свойство поместить в основной паттерн, а не в OPTIONAL, то в результат попадут только те объекты, у которых значение свойства установлено.

Результат выполнения этого запроса будет таким:

?name	?parent_name
"Alpha, JSC"	
"Beta, LLC"	"Alpha, JSC"

В семантических моделях каждое свойство может иметь любое количество значений – то есть, у каждой компании в нашем примере может быть несколько

вышестоящих организаций. Представим, что к компании Beta есть еще одно вышестоящее лицо, Gamma. Тогда результат запроса будет выглядеть так:

?name	?parent_name
"Alpha, JSC"	
"Beta, LLC"	"Alpha, JSC"
"Beta, LLC"	"Gamma"

Как мы видим, *для каждой комбинации вершин и ребер графов, отвечающих результатам запроса, выводится отдельная строка результата*. Таким образом, одна и та же сущность, одна и та же часть графа может фигурировать в результатах запроса несколько раз. Чем сложнее запрос, тем серьезнее будет его перегруженность повторяющимися результатами. Облегчить жизнь помогает выражение SELECT DISTINCT, выдающее только уникальные результаты запроса. Полезно также сокращать число возвращаемых запросом переменных.

Чтобы лучше понять механизм выполнения SPARQL-запроса, посмотрим на то, как содержащийся в запросе паттерн применяется к вершинам и ребрам графа – на примере последнего запроса:

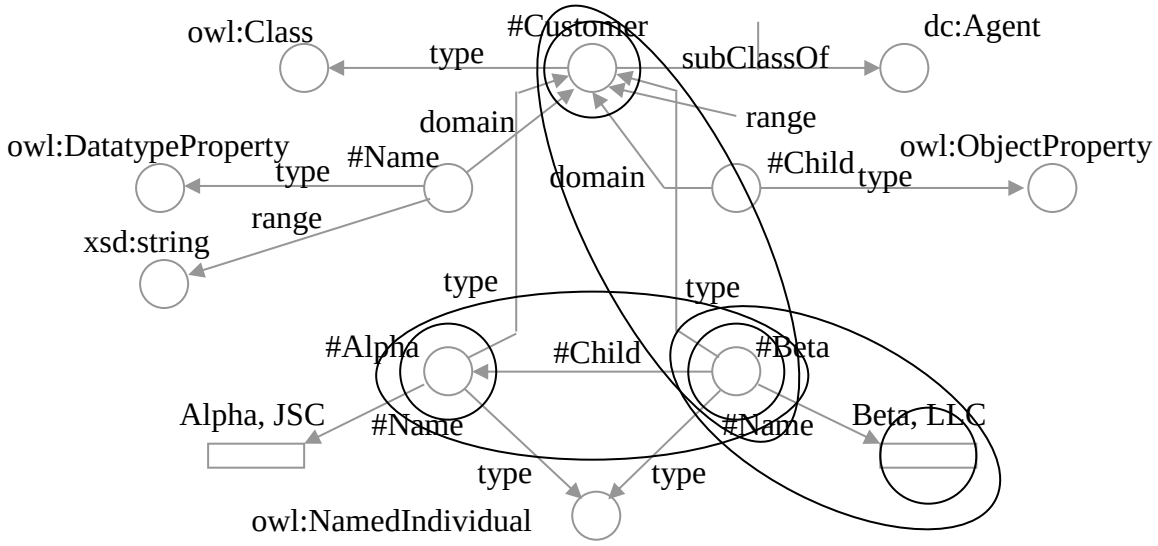

Рис. 21. Выполнение SPARQL-запроса – поиск ребер графа, отвечающих паттернам

На рис. 21 показана только одна часть графа, соответствующая этому запросу. Однако легко себе представить механизм выполнения SPARQL-запроса, как

«примерку» паттерна ко всем возможным комбинациям триплетов, существующим в модели.

К более глубокому рассмотрению синтаксиса SPARQL мы еще вернемся, а пока дадим краткий обзор некоторых полезных возможностей этого языка. Отсортируем результаты запроса на список всех клиентов по названию клиента, и выведем 10 результатов, начиная с 5-го:

```
SELECT * WHERE { ?cust rdfs:type <http://example.com/#Customer>.
                 ?cust <http://example.com/#Name> ?name }
                 ORDER BY ASC(?name) OFFSET 5 LIMIT 10
```

Результаты запроса можно фильтровать при помощи клаузы FILTER. Например, нам интересны только те клиенты, в названии которых содержится выражение LLC:

```
SELECT * WHERE { ?cust rdfs:type <http://example.com/#Customer>.
                 ?cust <http://example.com/#Name> ?name
                 FILTER(CONTAINS(STR(?name), "LLC")) }
```

При фильтрации можно сравнивать числовые значения, выполнять математические операции, но при этом необходимо следить за типами – чаще всего их нужно задавать в явном виде, используя знаки ^^. Обратите внимание, что в данном случае переменная ?name явно приводится к строковому типу.

Завершим рассмотрение темы SPARQL примерами запросов на добавление и удаление данных в онтологии. Запрос на добавление данных выглядит просто:

```
INSERT DATA { <http://example.com/#alpha> <http://example.com/#Name> "Alpha, JSC".
<http://example.com/#beta> <http://example.com/#Name> "Beta, LLC" }
```

Аналога SQL-запроса UPDATE в SPARQL нет, данные можно только удалять или добавлять. Поэтому, если возникает необходимость внести изменения в уже существующие триплеты, существующие данные нужно сначала удалить:

Вопрос для размышления

Почему в SPARQL нет аналога SQL-запроса UPDATE?

```
DELETE WHERE { <http://example.com/#beta> ?b ?c }
```

Этот запрос удалит всю информацию о клиенте Beta (но не удалит ссылки на него с других объектов). По понятным причинам, в запросе INSERT не могут использоваться переменные, а в запросе DELETE WHERE – могут.

Существует также запрос CONSTRUCT, аналогичный SELECT, но возвращающий результаты не в виде таблицы, а в виде нового RDF-графа (фактически – в виде файла).

Проверить существование решений у SPARQL-паттерна можно при помощи запроса ASK, возвращающего true, если есть хотя бы одно решение, и false в ином случае. Например, узнаем, существуют ли компании, имеющие родительскую организацию:

```
ASK { ?cust <http://example.com/#Child> ?parent }
```

На нашей модели результатом такого запроса будет true, истина.

В этой главе мы рассмотрели обычный режим работы с хранилищем триплетов посредством точки доступа SPARQL. В этом режиме точка работает подобно обычной базе данных, то есть возвращает ровно ту информацию, которая была в нее помещена.

Если рассматривать SPARQL и хранилища триплетов как функциональный аналог реляционной базы данных, то им здорово не хватает таких возможностей, как создание хранимых процедур и триггеров, контроля прав доступа к содержимому модели. «Родных» средств для решения этих задач в стандартах, действительно, нет. Однако мы успешно реализовали такую функциональность в промежуточном слое, через который приложения работают с моделью.

Между тем, мощь семантических технологий заключается в возможности автоматического получения логических выводов.

4.2. Машины и правила логического вывода

Машины логического вывода (англ. Reasoner) – еще один класс программного обеспечения, предназначенный для работы с онтологическими моделями. Они позволяют вычислять значения логических выражений, проверять правильность онтологии и автоматически помещать в нее новую информацию в соответствии с правилами. Машины логического вывода позволяют оперировать именами классов, свойств и сущностей, и «задавать модели вопросы», абстрагируя пользователя от подробностей внутреннего строения модели. В доступной пользователю форме проще всего познакомиться с Reasoner'ом Pellet, который можно установить как плагин к редактору Protégé, или использовать в качестве самостоятельного продукта. Популярными альтернативами являются HermiT и FaCT++.

Посмотрим, как машина логического вывода проверяет корректность онтологии. Создадим простую онтологию, состоящую из четырех классов:

Organization

    Company

    Institution

Person

В этой онтологии мы описали организации, которые делятся на компании и учреждения, а также персон – физических лиц. Зададим условие разобщенности (disjoint) для классов Company и Institution, поскольку ни одна компания не может являться одновременно и учреждением. Попробуем создать экземпляр класса Company, а затем добавить ему в список типов класс Institution. Reasoner выдаст сообщение об ошибке:

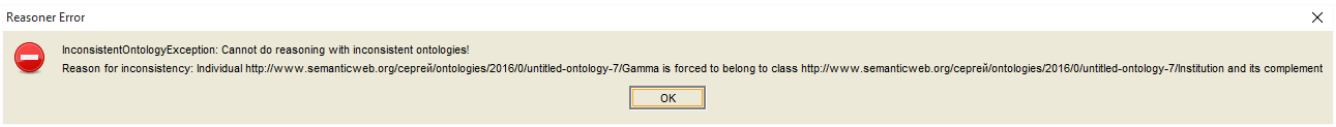

Рис. 22. Логическая ошибка в онтологии

Кроме того, знак ошибки появится в правом верхнем углу окна редактора Protégé:

Рис. 23. Знак логической ошибки в онтологии в редакторе Protégé.

При нажатии на него мы также сможем увидеть описание ошибки. Наконец, в списке типов нашей сущности оба класса будут подсвечены красным:

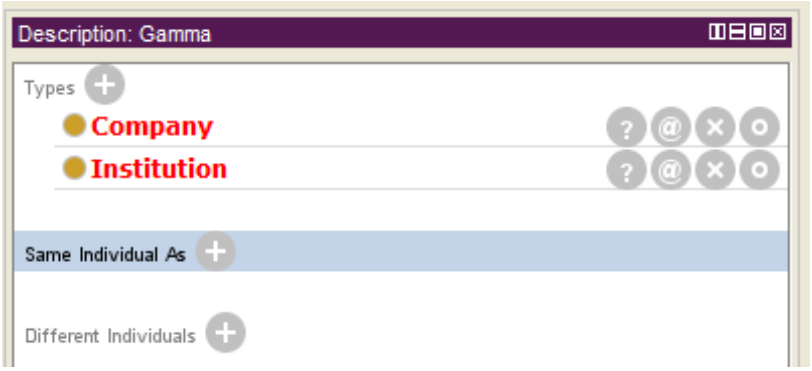

Рис 24. Неверно заданные классы в редакторе Protégé.

Теперь попробуем функционал получения логических выводов. Создадим свойство-ссылку worksIn (работает в), которая соединяет персон и организации, и объявим его функциональным инверсным свойством. Таким же образом создадим свойство hasEmployee (имеет сотрудника), которое соединяет организации с персонами. Укажем, что эти свойства являются инверсными друг для друга:

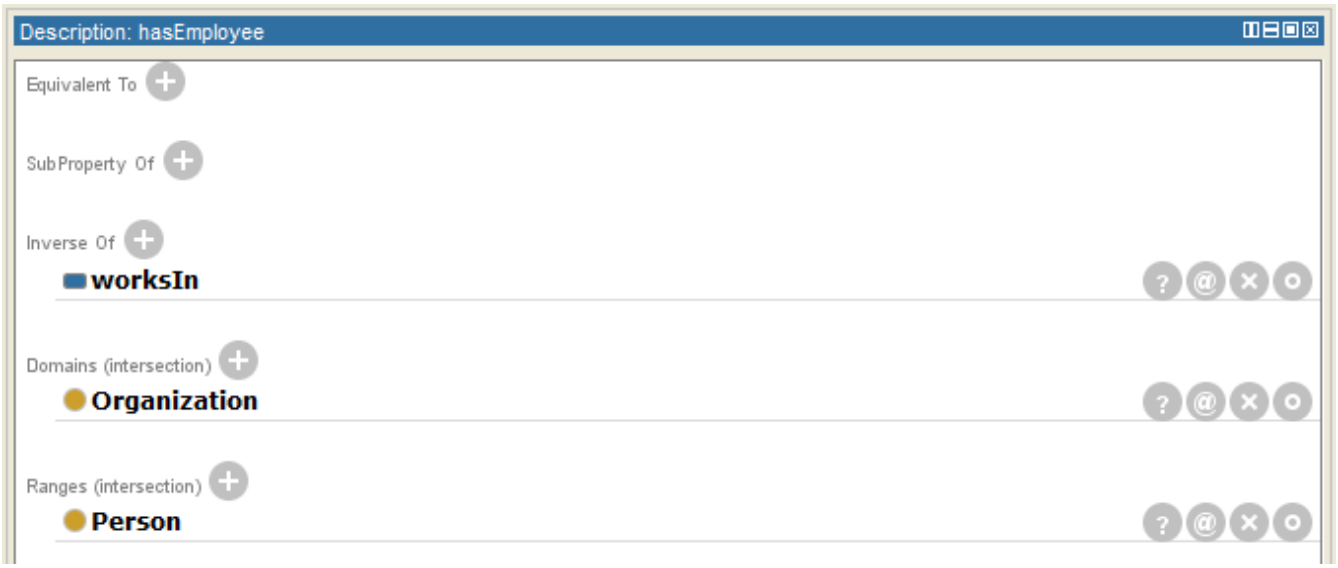

Рис. 25. Создание инверсного свойства

Теперь создадим индивидуальный объект класса Person под названием Smith, и укажем, что он работает в компании Alpha. Перейдя в свойства компании Alpha, увидим сделанный вывод, который состоит в том, что эта компания имеет сотрудника Smith:

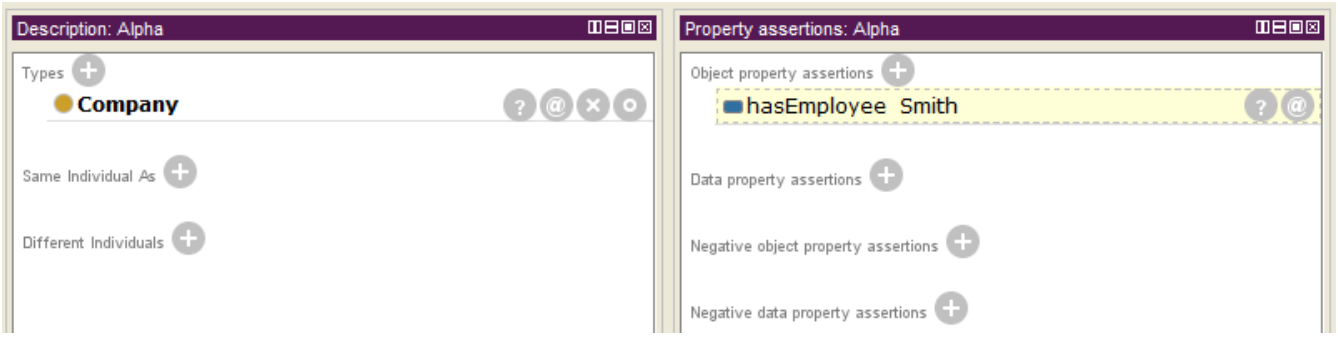

Рис. 26. Значение свойства, полученное в результате логического вывода

Перейдем к рассмотрению правил логического вывода. Каждый из стандартов онтологического представления информации содержит набор «корневых» аксиом. Например, для стандарта RDFS верна следующая аксиома:

```
rdf:type rdfs:range rdfs:Class .
```

Это выражение говорит о том, что областью значений предиката rdf:type является rdfs:Class. Таким образом, стандарты определяют свое содержимое через собственные термины.

Правила, по которым работают машины логического вывода, также приведены в стандартах. RDFS определяет базовый набор правил вывода, OWL существенно их расширяет. Среди правил логического вывода, определенных в RDFS, присутствуют следующие:

ЕСЛИ	**ТО**
aaa rdfs:domain xxx . yyy aaa zzz .	yyy rdf:type xxx .
xxx rdfs:subClassOf yyy . yyy rdfs:subClassOf zzz .	xxx rdfs:subClassOf zzz .
aaa rdfs:range xxx . yyy aaa zzz .	zzz rdf:type xxx .

Полный список правил логического вывода RDFS можно найти по ссылке http://www.w3.org/TR/2014/REC-rdf11-mt-20140225/#patterns-of-rdfs-entailment-informative – их не так много, всего 13. Выше мы привели в пример только некоторые из них, чтобы проиллюстрировать принцип действия машин логического вывода и пределы их возможностей. Например, согласно правилу, записанному во второй строке таблицы, если у класса xxx есть подкласс yyy, а у того – подкласс zzz, то считается, что xxx имеет подкласс zzz (транзитивность отношения «является подклассом». Это очень важно при практическом использовании онтологий, когда иерархии классов могут иметь сложную структуру.

Можно использовать логический вывод и при работе через точку доступа SPARQL – для этого нужно включить у нее entailment режим, соответствующий нужному стандарту. Тогда в ответ на все запросы точка доступа начнет возвращать не только триплеты, непосредственно содержащиеся в хранилище, но и триплеты,

полученные в результате логического вывода. Подробнее работа с entailment режимами будет рассмотрена в главе 6.

4.3. Правила логического вывода SWRL

Как мы видели в предыдущей главе, возможности логического вывода «по умолчанию» довольно скромны. Куда интереснее с практической точки зрения получить возможность определять собственные правила, в соответствии с которыми должны получаться логические выводы. Такая технология существует и называется SWRL. К сожалению, программные реализации SWRL далеки от промышленного уровня, хотя определенных результатов с их помощью можно достичь.

Синтаксис правил довольно прост. Каждое правило состоит из двух частей – условия и вывода, который формируется, если условие выполнено. И условие, и вывод могут состоять из нескольких *атомов* – элементарных логических выражений. Условие выполняется, если истинны все составляющие его атомы; если условие выполняется, то «сработают» все атомы, составляющие вывод. Внимание – отрицание и дизъюнкция в атомах не поддерживаются. Кроме того, благодаря парадигме «открытого мира» правила иногда не делают выводов, которые ожидает пользователь.

Каждый атом представляет собой предикат – утверждение о каких-либо объектах онтологии. Синтаксис записи предикатов соответствует принятому в логике. Например, если известно, что Петр является сыном Ивана, это может быть записано как бинарный предикат: являетсяСыном (Петр, Иван). То, что указано в скобках – аргументы предиката. В принципе, такая запись эквивалентна триплету

```
Петр являетсяСыном Иван
```

Унарный предикат имеет один аргумент и утверждает, что он является объектом, относящимся к определенному классу. Например, запись Человек (Иван) эквивалентна триплету

```
Иван  rdf:type    Человек
```

При составлении правил SWRL в качестве аргументов могут выступать переменные или конкретные объекты.

Предикаты могут представлять собой не только утверждения о связях; рассмотрим все варианты предикатов, которые могут использоваться в правилах SWRL.

1. Утверждения о принадлежности к классу – унарный предикат, аргументом в котором является индивидуальный объект либо переменная, его обозначающая. Простейшее SWRL-правило может выглядеть так:

   Мужчина (?x) -> Человек (?x)

   Это правило утверждает, что любой экземпляр класса Мужчина является и экземпляром класса Человек. Впрочем, столь простой вывод можно получить и средствами OWL без использования SWRL. Как мы видим, переменные в SWRL, как и в SPARQL, предваряются знаком вопроса; знак -> служит для разделения условий и вывода.
2. Утверждения о существовании связи в виде бинарного предиката. Примером является уже приводившееся выражение являетсяСыном (Петр, ?x).
3. Утверждения о значении свойства-литерала. При помощи способа, аналогичного приведенному в предыдущем пункте, можно получать в переменные значения свойств-литералов, и затем сравнивать их с различными выражениями при помощи встроенных функций. Пример: имеетВозраст (?x , ?y) ∧ swrlb:greaterThanOrEqual (?y, 16). Это условие проверяет, что объект ?x имеет возраст, больший или равный 16. Как мы видим из этого примера, знак ∧ используется для перечисления нескольких условий, соединенных логическим И – это и некоторые другие обозначения соответствуют общепринятой нотации для записи логических выражений (логическое ИЛИ в правилах SWRL не поддерживается). В Protégé версий 4 и 5 вместо этого знака нужно вводить запятую. Заметим

также, что для проверки равенства значения свойства какой-либо величине можно обойтись более коротким выражением: имеетИмя (?x, "Иван"), или являетсяСтудентом (?x, true).

4. Условие раздельности двух индивидуальных объектов: differentFrom (?x, ?y). Внимание: в Protégé 5 это выражение нужно писать с большой буквы, DifferentFrom.
5. Условие совпадения двух объектов: sameAs (?x, ?y). Условие будет истинно, если переменные ?x и ?y оказались равны одному и тому же объекту.
6. Условие о принадлежности значения переменной определенному типу данных. Например: xsd:int (?x)
7. Так называемые встроенные функции (SWRL built-ins), одно из которых мы уже видели в пункте 3: swrlb:greaterThanOrEqual. Имеется обширный набор таких функций, а также возможность определять собственные. Полный перечень таких функций можно найти в интернете – он включает выражения для работы с различными литеральными значениями, включая числа, даты и строки. Внимание: в Protégé 5 имена встроенных функций нужно вводить без префикса swrlb. Есть и другие пакеты встроенных функций – например, tbox, позволяющий проверять принадлежность сущностей стандартным классам RDFS/OWL.

Результат формулируется при помощи атомов в такой же форме, как и условие.

Рассмотрим пример создания и выполнения правила SWRL в редакторе Protégé. Его старая версия, Protégé 3, имела визуальный интерфейс для построения правил. Окно ввода правила в ней выглядит так:

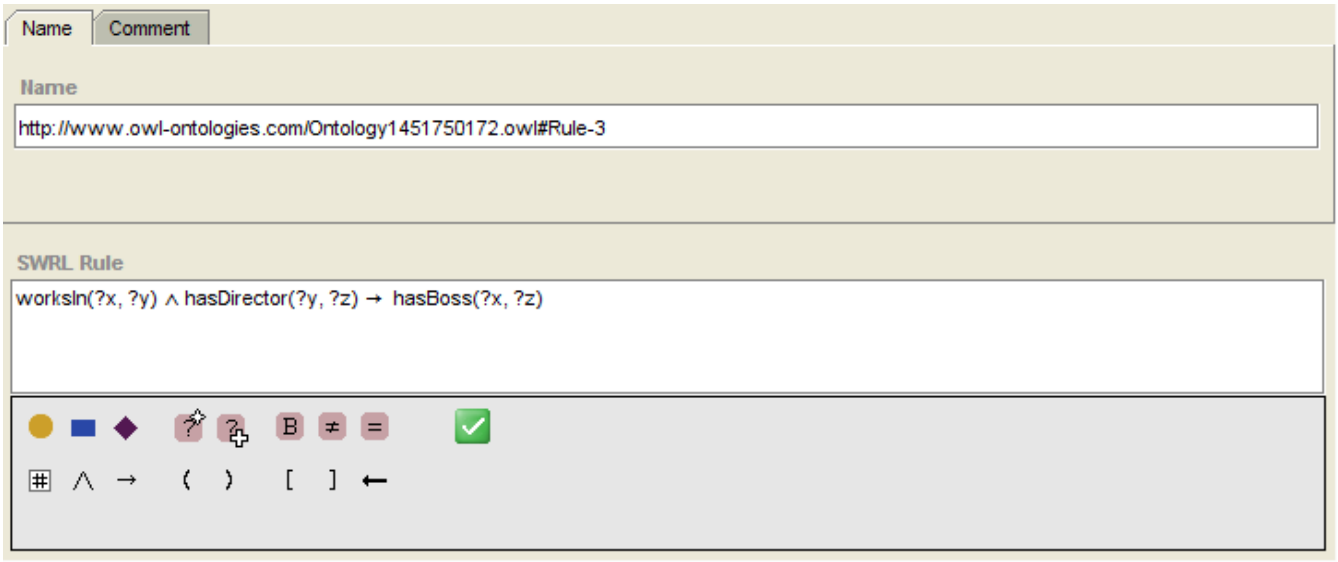

Рис. 27. Создание SWRL-правила в редакторе Protégé 3.

Это правило гласит, что если ?x работает в компании ?y, а компания ?y имеет директора ?z, то ?x имеет начальника ?z. Редактор выдает подсказки в процессе ввода правил, подсвечивает неверно введенные правила.

В текущей версии редактора, Protégé 5, визуального конструктора правил нет. Правила вводятся на закладке SWRL в текстовом виде. Добавление правила происходит в диалоговом окне после нажатия кнопки «+»:

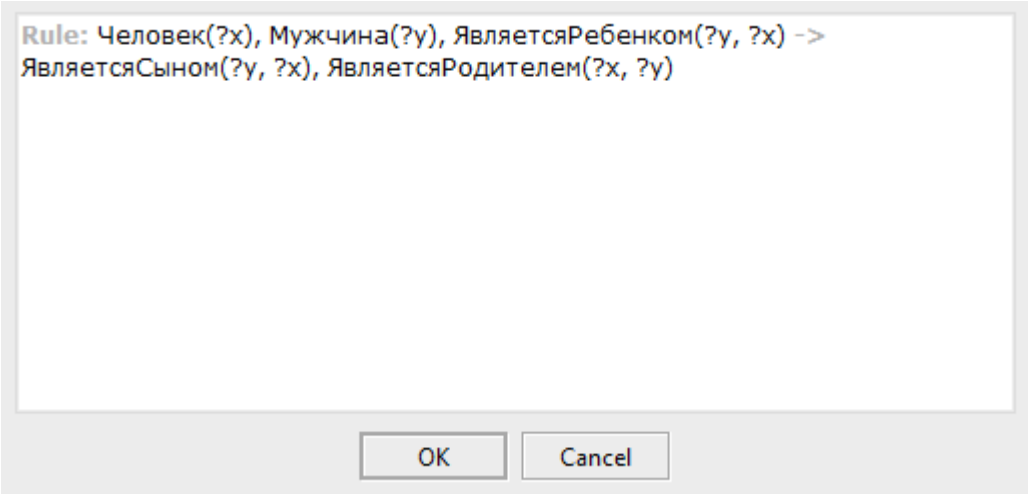

Рис. 28. Создание правила SWRL в редакторе Protégé 5.

Если правило успешно создано, оно появится в списке:

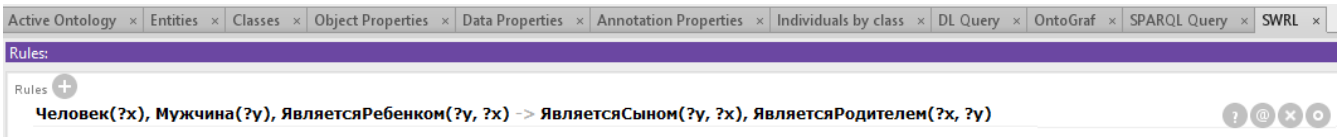

Рис. 29. Список правил в редакторе Protégé 5.

К сожалению, в Protégé полностью отсутствуют средства отладки правил, и даже сообщения об ошибках их синтаксиса. Если при создании правила что-то пойдет не так, можно будет увидеть только Java-исключение, нажав на уже описанную нами выше кнопку в виде красного треугольника в правом верхнем углу окна приложения.

Вернемся к примеру онтологии из прошлой главы. Создав в онтологии персону Jackson, связь hasDirector (Alpha, Jackson), и применив наше правило, увидим в свойствах пользователя Smith, что его боссом является Jackson.

Механизм применения правил в Protégé 3 достаточно непрост, поскольку для этого используется экспорт правил и всего содержимого онтологии в компонент исполнения бизнес-правил Drools, применение правил, а затем их импорт в обратном направлении. В Protégé 4 и 5 для выполнения правил SWRL используется машина логического вывода, напрямую работающая с содержимым онтологий, зато отсутствует визуальный интерфейс для создания правил.

Как мы видим, правила SWRL позволяют создавать весьма гибкие условия для получения новых знаний. Недостатком этой технологии является необходимость специальной подготовки пользователя для конструирования правил, наличие у него глубоких знаний онтологии, а также слабые программные реализации.

4.4. OWL API

OWL API – программный интерфейс, позволяющий работать с хранилищами триплетов и машинами логического вывода из Java-приложений. Большинство

инфраструктурного ПО для работы с семантическими технологиями реализовано как раз на Java, поэтому данный способ является в значительной степени «родным». Например, именно его использует редактор Protégé. Недостатком такого подхода является то, что логика работы с онтологической моделью оказывается запрограммированной в коде, который требует перекомпиляции при изменении. Следовательно, при серьезном изменении модели или алгоритма работы с ней может потребоваться вмешательство программиста, что не рационально с точки зрения владельца информационной системы. Кроме того, существует ряд технических проблем – например, с некоторыми хранилищами триплетов нельзя работать одновременно более чем из одной Java-машины.

Достоинствами OWL API является высокий уровень гибкости и доступности «продвинутых» возможностей инфраструктурного ПО. Например, при доступе через SPARQL точку доступа нет возможности отличить «выведенные» факты от исходно содержавшихся в модели, а при доступе через OWL API – есть.

В целом, мы рекомендуем внимательно проанализировать необходимость применения именно этого способа работы с семантическими моделями из прикладного ПО перед принятием решения о его использовании. Аргументом может стать необходимость выполнять операции, недоступные при других способах работы с моделью.

5. Прикладное ПО для работы с семантическими моделями

Рассмотренные нами выше редактор Protégé, хранилища триплетов и точки доступа SPARQL, OWL API являются инструментами для разработчиков. В этом разделе мы познакомимся с несколькими приложениями, которые дают возможность работать с семантическими моделями пользователям, не имеющим глубоких познаний в технической стороне онтологического моделирования.

5.1. Редактор онтологий Onto.pro

Onto.pro – онлайн-редактор онтологий, созданный компанией «ТриниДата». Его онлайн-версия доступна по адресу http://onto.pro. Редактор предоставляет функционал создания простых семантических моделей. Его отличительной особенностью является работа с содержимым хранилища триплетов через SPARQL точку доступа – то есть, редактор не использует файлы для хранения онтологий, а работает напрямую «с базой».

Для разграничения разных онтологий в Onto.pro используется понятие «точка доступа», которое соответствует не только физическому хранилищу онтологии, но и ее корневому элементу – объекту, от которого порождены все остальные элементы онтологии. Это сделано для изоляции друг от друга данных разных пользователей, работающих с одним и тем же хранилищем (набором данных в triple store).

Создание классов

Для удобства восприятия, классы в Onto.pro представлены в виде иерархии, основанной на отношениях «является подклассом» (хотя на самом деле настоящей иерархии у классов может и не быть; иерархия является лишь одним, частным случаем организации взаимоотношений классов – так, один класс может быть подклассом сразу нескольких классов).

После входа в только что созданную точку доступа (онтологию), дерево слева пусто – объектов в нем еще нет:

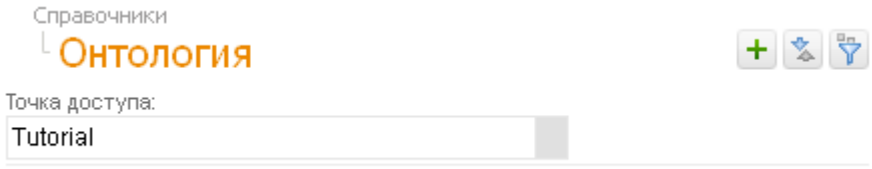

Рис. 30. Выбор точки доступа

Чтобы создать новый класс, нажмем кнопку «+» на панели инструментов (кнопка показана на рисунке выше). Справа появится форма создания класса, в которой присутствуют поля, соответствующий некоторым из определенных в RDF/OWL стандартных свойств. Ниже, под формой, можно увидеть интерфейс для выбора надкласса создаваемого класса. Сама же форма создания выглядит так:

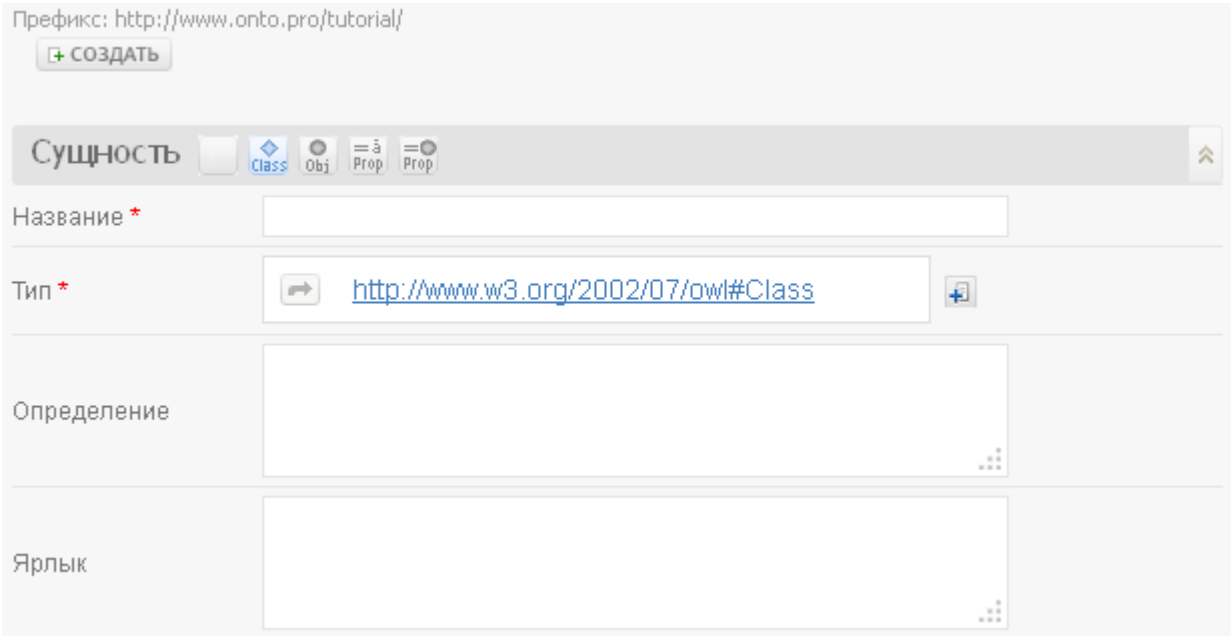

Рис. 31. Форма создания сущности

Обратите внимание на набор кнопок справа от заголовка формы «Сущность». С его помощью можно выбрать тип создаваемого элемента:

Сущность без типа (точнее, с типом Thing, который применим к любым элементам OWL);

- Класс;
- Индивидуальный объект;
- Свойство-литерал;
- Свойство-указатель на объект.

Нам необходимо создать класс, поэтому активна вторая кнопка переключателя.

Введем в поля «Название» и «Ярлык» значение «Организация». Название – это завершающая часть уникального идентификатора объекта (URI). Она не должна содержать пробелов и спецсимволов, поэтому при сохранении записи они будут отфильтрованы. Ярлык – это читаемое название сущности (label), которое будет использоваться при ее выборе в редакторе.

После нажатия кнопки «Создать» класс «Организация» появится в дереве слева. Теперь нам нужно создать его подкласс – «Компания». Для этого нажмем кнопку «+» на панели инструментов еще раз. Форма в середине страницы снова очистится, и мы сможем ввести название и ярлык для нового класса – «Компания». Обратим внимание на нижнюю часть формы:

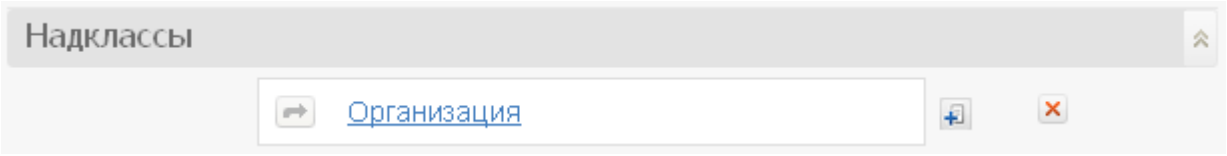

Рис. 32. Выбор надкласса

В области «Надклассы» указывается родительский класс для того класса, который мы создаем. По умолчанию здесь выбирается последний класс, который мы создавали или просматривали. Нажатием на красную кнопку «X» можно очистить выбранный класс – эта возможность нам понадобится, когда мы будем создавать класс «Персона», находящийся на верхнем уровне иерархии. Нажатие на синюю кнопку «+» открывает диалоговое окно выбора объекта – здесь можно выбрать любой другой надкласс для создаваемого класса:

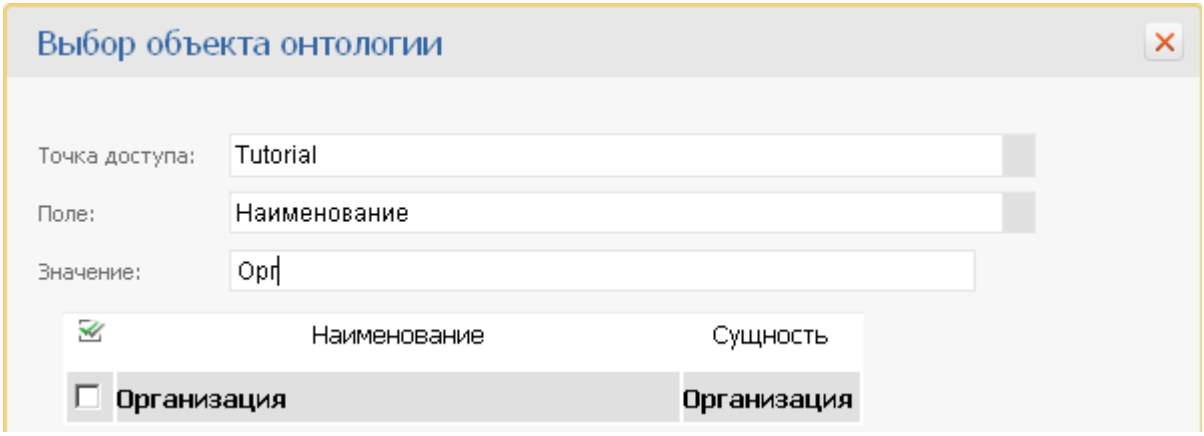

Рис. 33. Выбор объекта

В этом диалоговом окне можно быстро найти нужный элемент подбором по части названия. Щелчок на имени элемента выбирает его.

После того, как класс создан, в форме «Надклассы» справа появится зеленая кнопка «+», при помощи которой можно будет добавить еще надклассы для данного класса. Мы можем убедиться в этом, создав класс «Компания», и посмотрев в область «Надклассы».

После создания трех классов из нашего примера, дерево классов примет следующий вид:

Рис. 34. Иерархия из трех классов

Создание свойств-литералов

Чтобы иметь возможность выражать содержательные сведения о наших объектах, необходимо создать для них наборы свойств. Как мы уже говорили, свойства делятся на два типа: одни принимают значения-литералы (строки, числа и т.д.), другие предназначены для связывания разных объектов между собой.

Начнем с создания свойства-литерала. В дереве выберем класс, объекты которого будут обладать данным свойством. Переключимся на просмотр списка свойств-литералов, выбрав соответствующую кнопку над списком сущностей, связанных с этим классом (расположен ниже дерева):

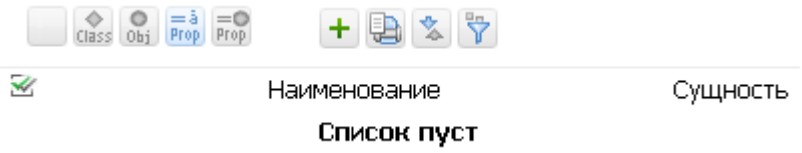

Рис. 35. Список экземпляров

После нажатия кнопки «+» в панели инструментов этого списка, убедимся, что в форме создания сущности выбран ее тип – «Свойство». Нам нужно создать свойство «Адрес»; заполненная форма его создания будет выглядеть так:

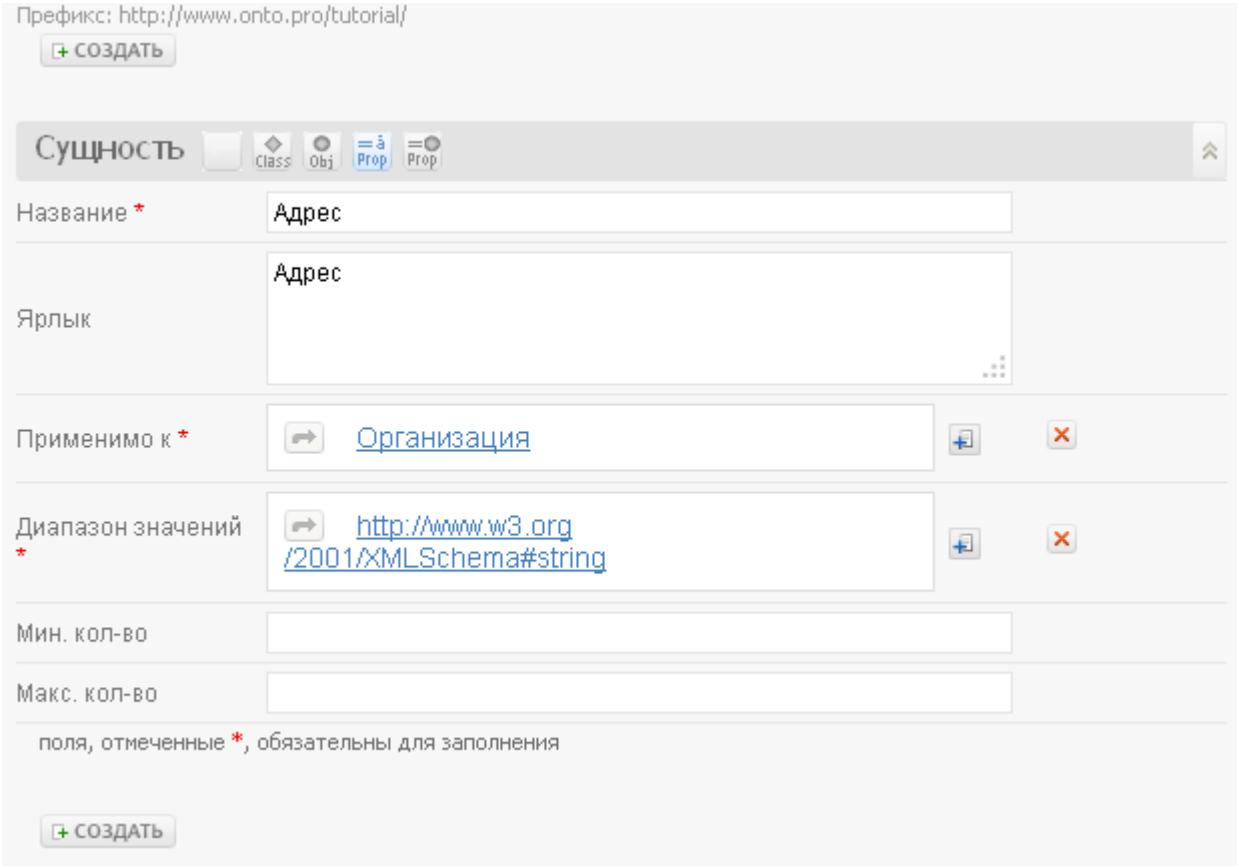

Рис. 36. Форма создания свойства

Обратите внимание, что значение в поле «Диапазон значений» мы выбрали при помощи диалогового окна, в котором отображается список типов данных RDF/OWL, поддерживаемых Onto.pro.

Еще одна важная особенность Onto.pro состоит в том, что он воспринимает классы, к которым применимо свойство, и диапазоны значений свойств как ограничения. Задать значение свойства можно будет только для индивидуальных объектов, входящих в класс, указанный в поле «Применимо к» – в отличие от описанного выше редактора Protégé.

Также в отличие от Protégé в настройках Onto.pro можно указать, как воспринимается указание нескольких значений в полях «Применимо к» и «Диапазон значений». В Protégé, как мы уже упоминали, при задании двух и более

классов в любом из этих полей результатом будет их пересечение. Например, если в поле «Применимо к» (domain) выбраны классы Контрагент и Юридическое лицо, будет считаться, что значением свойства могут обладать только индивиды, относящиеся к обоим этим двум классам одновременно. В Onto.pro есть возможность переключаться между пересечением и объединением классов: во втором случае обладать значением свойства смогут как индивидуальные объекты класса Контрагент, так и объекты класса Юридическое лицо. В триплетах это выражается путем создания класса-объединения, на который указывает значение свойства «Применимо к». То же самое относится к диапазонам значений свойств-связей.

По умолчанию в Onto.pro используется второй вариант: указание нескольких классов воспринимается как объединение. Продолжим работу с нашим тестовым свойством и укажем, что оно присуще не только объектам класса «Компания», но и персонам. Для этого нужно нажать зеленую кнопку «+» справа от первого значения поля «Применимо к». Появится вторая строка, для следующего экземпляра поля; нажмем на кнопку выбора значения, и в появившемся диалоговом окне выберем (или найдем по названию) класс «Персона». Результат будет выглядеть так:

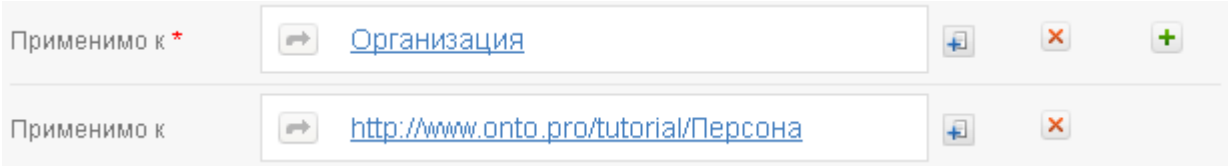

Рис. 37. Выбор нескольких диапазонов значений для свойства

Теперь форму нужно сохранить; после этого можно будет добавить еще один экземпляр значения свойства, и так далее.

Убедимся, что мы создали свойство, которое присуще объектам двух разных классов. Выберем в дереве класс «Организация», и установим переключатель типов сущностей в расположенном ниже списке в положение «Свойства». В списке появится наше свойство – «Адрес»:

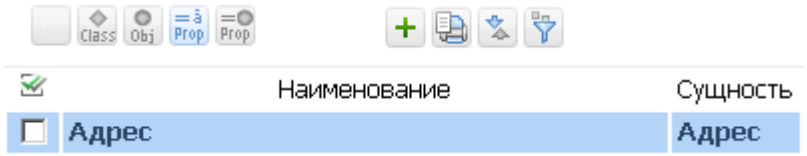

Рис. 38. Результат создания свойства

Выбрав в дереве класс «Персона», можно убедиться, что свойство «Адрес» действительно и для него. Теперь нужно создать второе свойство из нашего примера – «Дата рождения». Делается это по аналогии с созданием адреса, только в качестве типа значения надо выбрать «Дата», а в поле «Применимо к» оставить только одно значение – «Персона». Также, поскольку каждая персона имеет одну и только одну дату рождения, в поля «Мин. количество» и «Макс. количество» нужно ввести цифры 1. Эти поля указывают, какое количество значений может иметь данное свойство для каждого объекта. Возможность иметь более одного значения для каждого свойства – одно из отличий семантической информационной модели от реляционной.

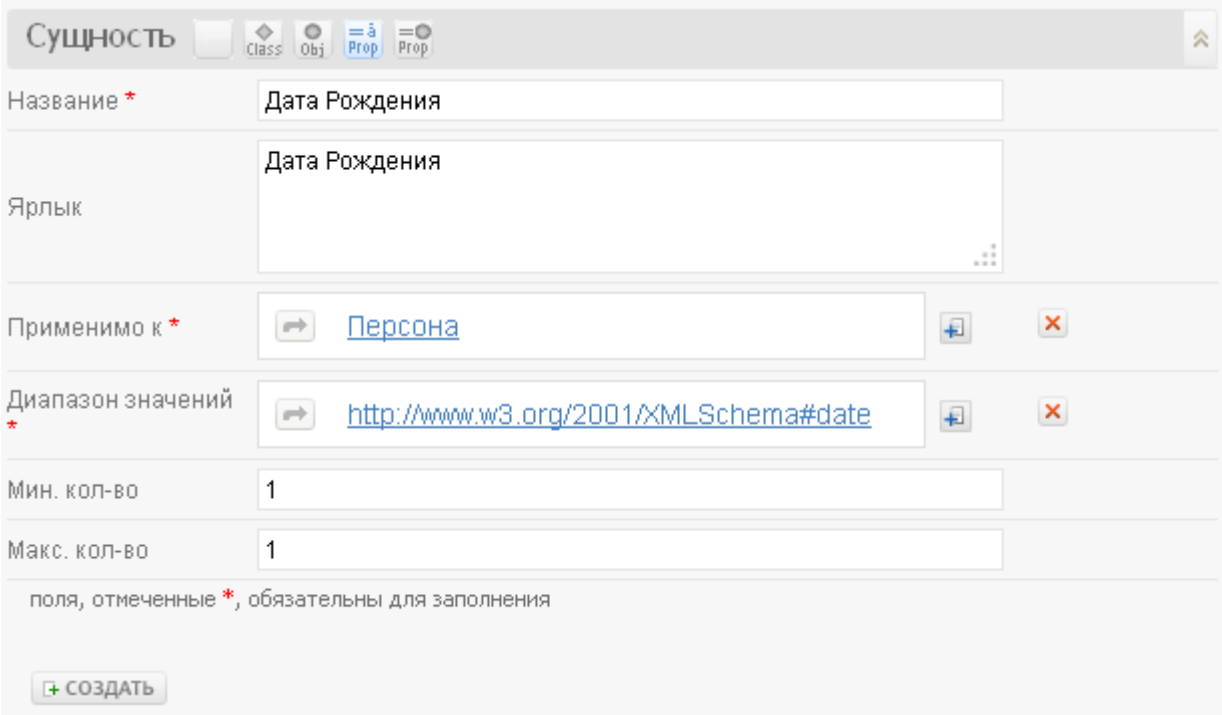

Рис. 39. Ограничения на количество значений для свойства

Свойства готовы. Заметим, что Onto.pro воспринимает ограничения именно как ограничения, и не даст присвоить свойству больше значений, чем в них задано.

Рассмотрим создание свойств-указателей на объекты, а затем перейдем к созданию экземпляров, и заполнению значений свойств.

Создание свойств-указателей на объекты

Свойства-указатели на объекты создаются точно также, как и обычные свойства; нужно только устанавливать переключатель типа сущности в положение «Свойство-указатель на объект».

Поле «Диапазон значений» в этом случае приобретает тип «выбор объекта». Здесь мы должны выбрать класс, на экземпляры которого будет ссылаться данное свойство. Выше мы уже говорили о том, что при выборе нескольких классов они

могут восприниматься как объединение или пересечение в зависимости от настроек редактора.

Пусть для нашего примера нам нужно создать свойство «Является учредителем». Для этого выбираем в дереве класс, объекты которого будут иметь это свойство – «Компания»; в списке внизу выбираем переключатель «Свойства-указатели на объект», и нажимаем кнопку «+» («Добавить»). Заполняем форму создания свойства:

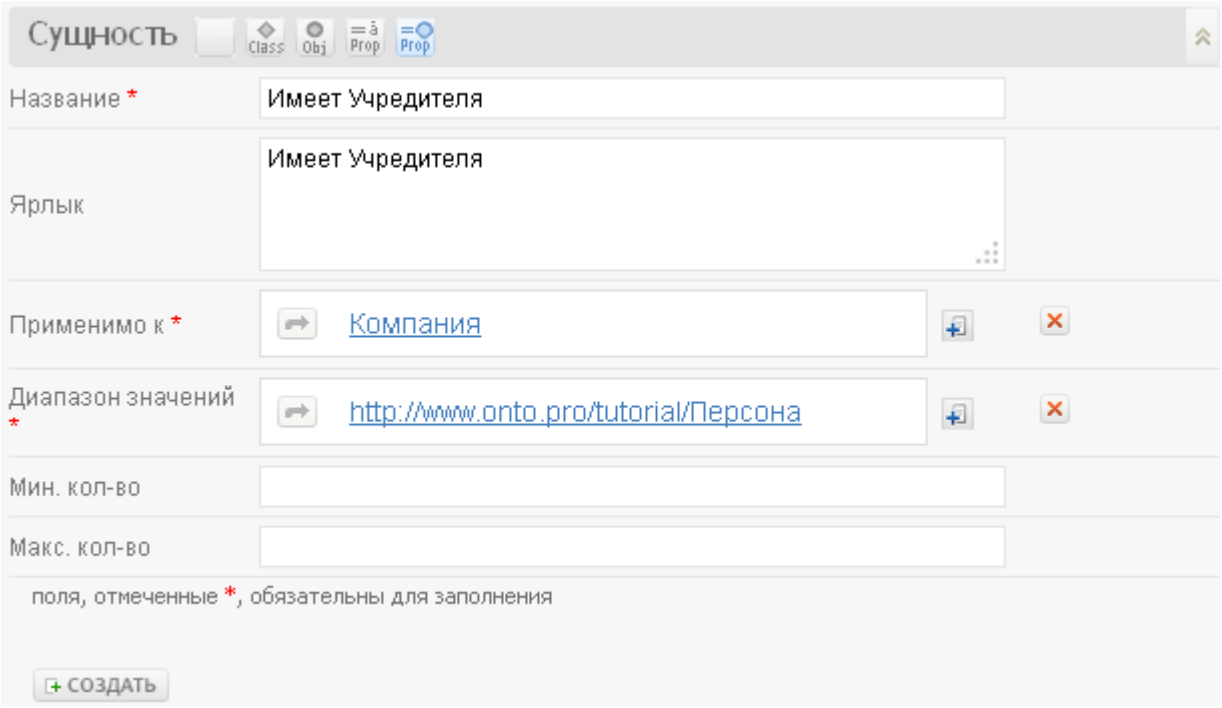

Рис. 40. Создание свойства-указателя на объект

Создание индивидуальных объектов

Перейдем к созданию индивидуальных объектов, относящихся к ранее созданным классам. Для этого выбираем класс-родитель в дереве, а затем нажимаем кнопку «+» внизу, над списком сущностей, относящихся к данному классу. Для начала, создадим персон. В средней части страницы появится форма, которую надо заполнить таким образом:

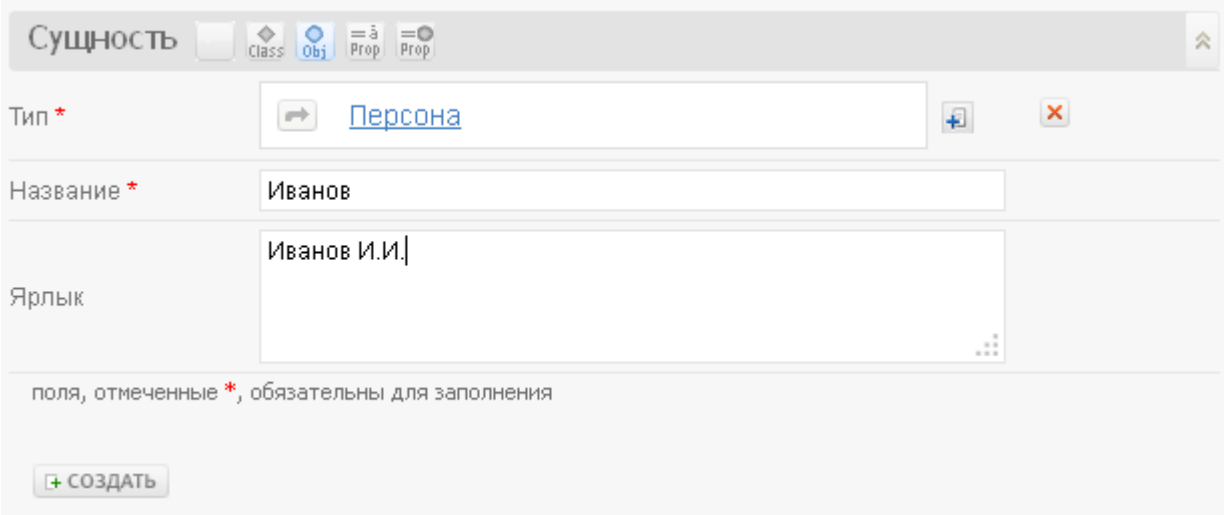

Рис. 41. Создание индивидуального объекта

Обратите внимание, что в заголовке формы активен переключатель типа сущности «Объект».

После нажатия кнопки «Создать», будет создан индивидуальный объект, который отобразится в списке объектов:

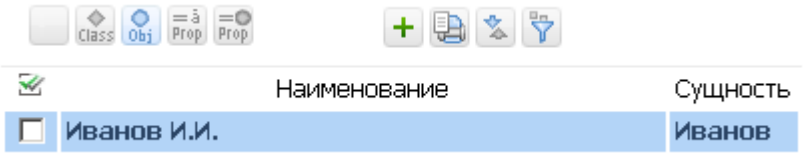

Рис. 42. Созданный объект в списке

Выбрав этот объект, мы сможем отредактировать все его свойства, написать к нему комментарии, прикрепить файлы, и посмотреть результат в wiki-представлении.

Обратим внимание на то, что объект имеет свойства, определенные для того класса, к которому он относится, а также унаследованные от родительских классов. Так, форма свойств объекта «Иванов» имеет следующий вид:

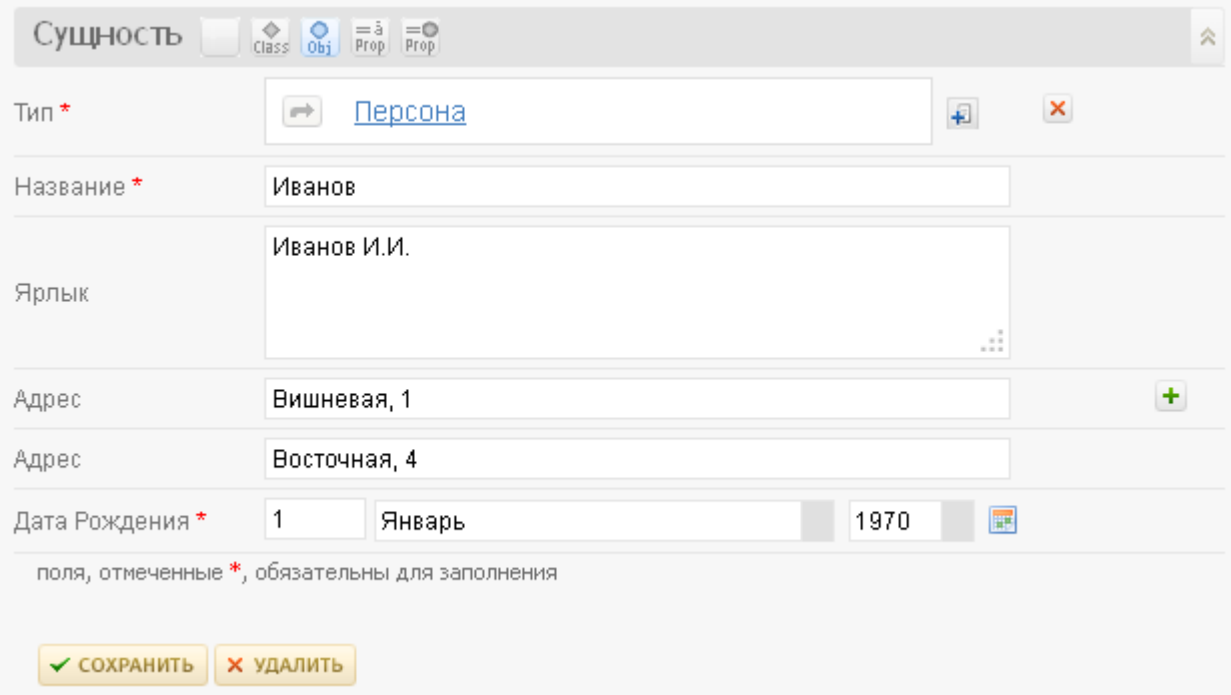

Рис. 43. Редактирование свойств объекта

Обратите внимание, что поле «Адрес» в этом примере имеет несколько значений. Добавить новые значения можно нажатием на зеленую кнопку «+», расположенную справа от поля.

Просмотр информации о классах в wiki-представлении

Нажав на ссылку «посмотреть в wiki-представлении», расположенную в верхней части формы редактирования свойств сущности, мы увидим следующую страницу (пример для ООО «Альфа» из онтологии, используемой в этом тестовом примере):

Рис. 44. Информация о сущности в вики-представлении.

Щелчком по ссылкам на любую сущность (например, на И.И. Иванова) можно перейти на соответствующую ей страницу, где будет отображена обратная ссылка, а также ссылки на другие элементы семантической модели. Эти страницы доступны неавторизованным пользователям Onto.pro; на ней также выводятся комментарии и файлы, прикрепленные в Onto.pro к данному элементу онтологии. Таким образом, создавая информационную модель, мы одновременно делаем ее доступной для просмотра неавторизованными пользователями, в виде wiki-представления. Пользователь также может искать термины в wiki, зайдя по адресу http://www.onto.pro/[имя онтологии], и введя интересующее слово в строке поиска. Больше возможностей работы с моделью предоставляет Система Управления Знаниями АрхиГраф.СУЗ.

Возможности Onto.pro также включают разграничение и контроль прав доступа пользователей к элементам модели (на уровне точек доступа и классов), импорт-экспорт содержимого онтологий через Excel, отслеживание и отображение истории редактирования онтологии.

5.2. Контролируемый естественный язык: редактор Fluent Editor

Одним из способов приближения семантических технологий к особенностям человеческого восприятия является использование контролируемого естественного языка (CNL, Controlled Natural Language). «Естественный язык» - это любой из человеческих языков: английский, русский и т.д. «Контролируемым» он называется потому, что на синтаксис языка налагаются определенные ограничения, чтобы облегчить интерпретацию языковых выражений в виде логических утверждений. Мы коротко рассмотрим возможности одного из программных продуктов для работы с CNL – редактор FluentEditor компании Cognitum. Этот редактор является частью Фреймворка Ontorion, в который входит также распределенное хранилище семантических моделей (аналог хранилища триплетов), построенное по собственной технологии, и веб-интерфейсы для доступа к нему.

Fluent Editor обладает возможностью импорта/экспорта моделей в формат OWL, работы с «облачным» хранилищем онтологий, что делает его пригодным для работы и в интегрированных средах семантического ПО.

Окно редактора состоит из трех основных частей: поля для ввода текста, где мы вводим логические выражения на обычном языке (используется английский), дерева получившейся онтологии в правой части, и окна машины логического вывода внизу. Попробуем решить с помощью FluentEditor небольшую практическую задачу.

Пусть мы имеем дело с моделью угроз, присущих им факторов, и устройств, позволяющих обнаружить те или иные факторы, и предотвратить угрозы. Часть модели, описывающая классы и типы отношений, будет выглядеть так:

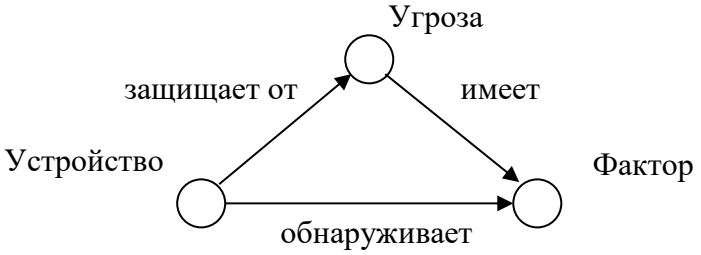

Рис. 45. Классы для модели угроз и защиты от них

Простое логическое правило, которое мы хотим выразить в модели, формулируется так: устройство защищает от угрозы, если оно обнаруживает ее фактор. Посмотрим теперь на часть модели, описывающей индивидуальные объекты и их свойства:

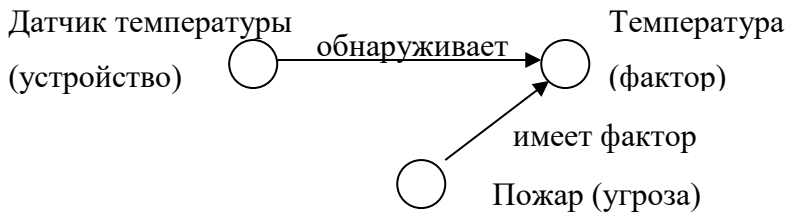

Рис. 46. Экземпляры для модели угроз и защиты от них

Как мы видим, устройство «датчик температуры» обнаруживает фактор «температура», который соответствует угрозе «пожар». В этой модели нет в явном виде информации о том, что датчик температуры позволяет предотвратить пожар – мы хотим получить ее в качестве логического вывода.

Начнем вводить соответствующие выражения во Fluent Editor’e. В зависимости от места в предложении и окружающих грамматических конструкций, редактор воспринимает те или иные термины как определения классов, свойств или индивидуальных объектов. Наша первая фраза:

```
Every defensive-equipment prevents nothing-but thing that is a threat.
```

Это выражение задает сразу два класса («защитное устройство» и «угроза»), и отношение между ними («предотвращает»). Теперь дадим определение фактора, и его связей с угрозой и оборудованием:

```
Every threat has-damage-factor a factor.
Every defensive-equipment detects a factor.
```

Теперь перейдем к экземплярам:

```
Humidity is a factor.
Temperature is a factor.

Fire is a threat. Flooding is a threat.
Fire has-damage-factor Temperature. Flooding has-damage-factor Humidity.

Heatdetector is a defensive-equipment.
Heatdetector detects Temperature.
```

Мы задали все необходимые сведения. Осталось только придумать способ получения логического вывода, и задать для него правила. Создадим синтетическое свойство «поддается обнаружению датчиком температуры», и определим правило, по которому оно присваивается объектам:

```
Something is a detectable-by-heat-detector if-and-only-if-it is a threat that has-damage-factor that is detected by Heatdetector .
```

Мы утверждаем, что нечто подлежит обнаружению датчиком температуры в том и только в том случае, если оно является угрозой, имеющей фактор, который обнаруживается датчиком температуры. Изъян у этой конструкции только один – в ней явно указан датчик оборудования. К сожалению, текущий синтаксис CNL не позволяет задать условие в общем виде – что «оборудование защищает от угрозы, если она имеет фактор, обнаруживаемый этим оборудованием».

Теперь остается только задать вопрос Reasoner’у (во FluentEditor’е, как и в Protégé, используется HermiT, но интерфейс построения запросов к нему лучше развит):

```
Who-or-what is a detectable-by-heat-detector ?
```

Машина логического вывода сообщает нам ответ: Fire.

Вот как это выглядит в окне FluentEditor:

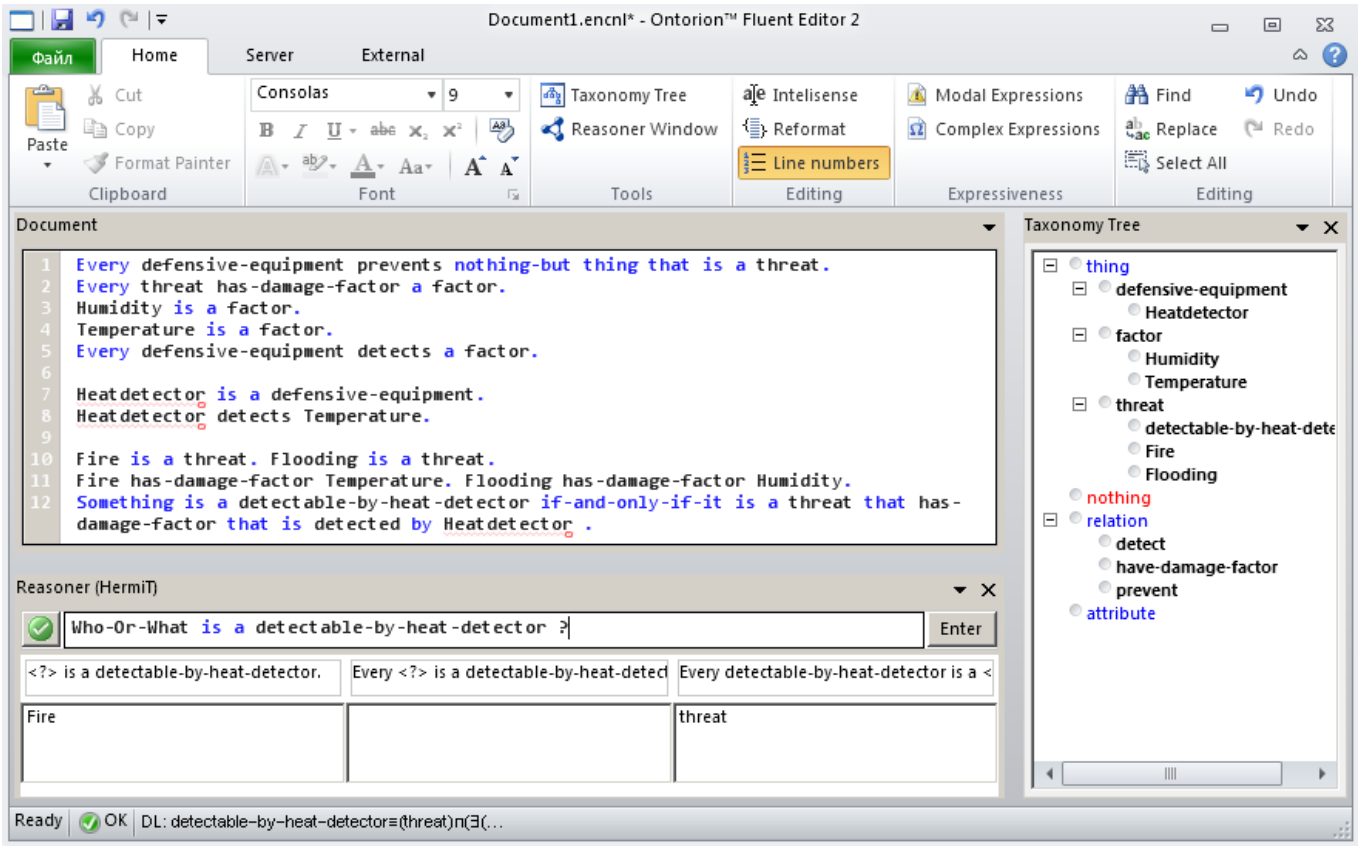

Рис. 47. Приложение FluentEditor

Модель можно построить немного по-иному, «перевернув» свойство have-damage-factor. Пусть не угроза имеет фактор, а фактор сигнализирует об угрозе. Тогда последнее логическое выражение в модели можно будет сформулировать по-другому:

```
Something prevents Fire if-and-only-if-it detects a factor that signals Fire.
```

Это не избавило нас от указания конкретного объекта в утверждении, но позволило не вводить лишний тип свойства для получения результата. Теперь мы можем задать модели такой вопрос:

```
Who-or-what prevents Fire?
```

Ответом будет Heatdetector.

Средства контролируемого логического языка находятся в начале своего развития, однако это направление очень перспективно с точки зрения широкого охвата семантическими технологиями различных сфер применения. Напомним, что получаемые в редакторе модели можно экспортировать в OWL, и наоборот. Таким образом, с логическими выражениями в редакторе сможет работать человек, не знакомый с технологическими средствами семантического моделирования, которые мы рассматривали в предыдущем разделе. В процессе ввода выражений редактор «подсказывает» синтаксис, и выделяет ошибочно введенные выражения.

5.3. Запросы к онтологической модели: система АрхиГраф.СУЗ

Мы убедились, что большинство программных средств логического вывода пока недостаточно дружественны к пользователю. Перед разработчиками, желающими уже сегодня использовать преимущества онтологических моделей в продуктах, работать с которыми будут люди без специальной подготовки, стоит задача построения простых и понятных интерфейсов хотя бы для извлечения информации из моделей. Нашей собственной разработкой является система АрхиГраф.СУЗ, одной из главных функций которой является предоставление разнообразных возможностей поиска знаний в модели.

Возьмем в качестве примера несложную модель, описывающую курорты и расположенные недалеко от них достопримечательности. Пусть имеются следующие классы и отношения:

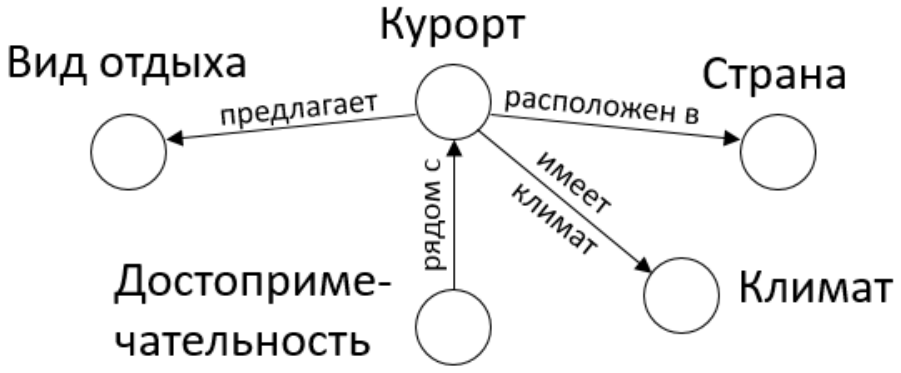

Рис. 48. Онтология для построения запросов

Индивидуальные объекты каждого класса обладают рядом свойств – названиями, датой назначения и т.д.

Начнем знакомство с АрхиГраф.СУЗ с самого простого режима – поиска по параметрам. В этом режиме система предлагает пользователю выбрать тип искомого объекта – пусть это будет «Курорт». После этого приложение выводит форму поиска, в которой можно задать значения всех или некоторых свойств, которыми могут обладать объекты искомого типа. Легко заметить, что этот режим похож на фасетный поиск в интернет-магазинах, где товары разных категорий обладают параметрами разных типов:

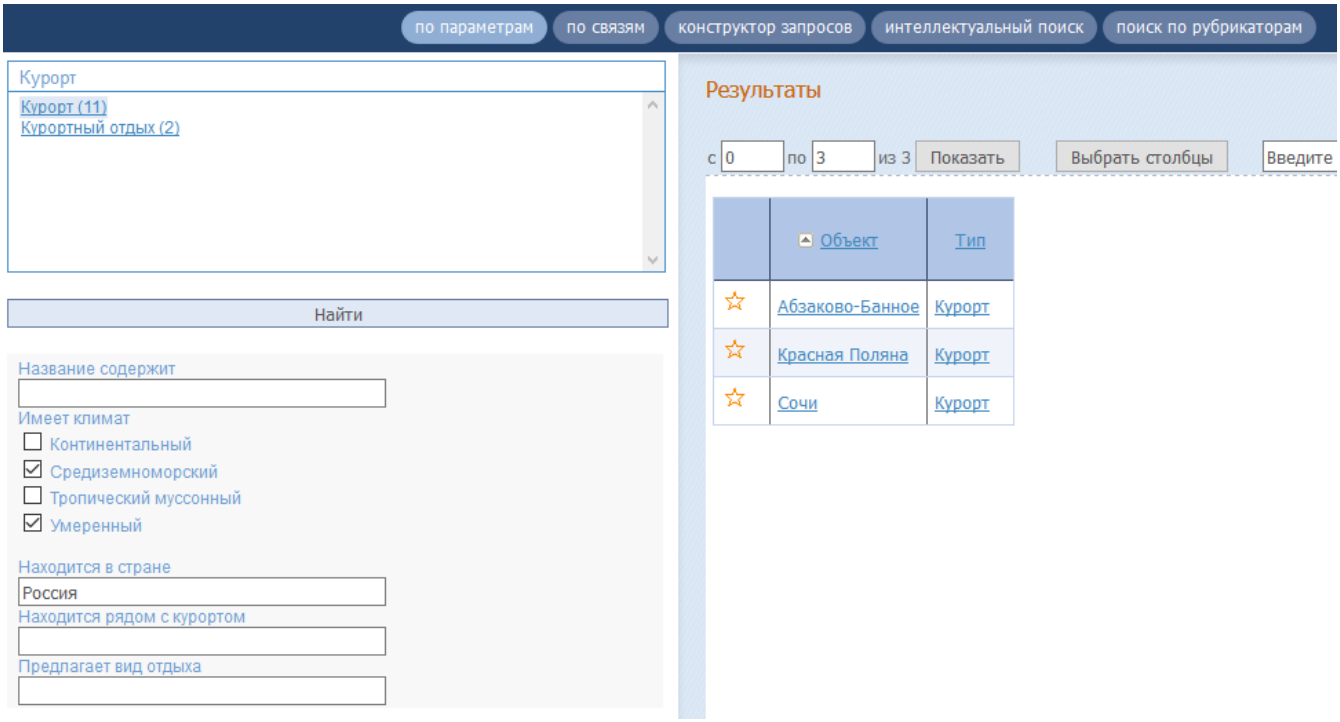

Рис. 49. Режим поиска по параметрам

После задания параметров и нажатия кнопки «Найти», справа мы увидим список искомых объектов. С ними можно выполнять разные операции: отсортировать, выбрать для отображения дополнительные столбцы, экспортировать список в Excel, открыть окно свойств каждого объекта. Можно сохранить и условия самого запроса.

Второй режим называется «поиск по связям». Он работает по принципу известного детского стихотворения «Дом, который построил Джек». Мастер пошагово задает вопросы, чтобы привести нас к решению задачи. Первый вопрос касается типа сущности, которую мы хотим найти. Предположим, мы хотим получить список всех стран, в которых есть курорты, где можно заняться дайвингом. Значит, искомым типом сущности будет «Страна».

Второй вопрос, который задаст нам мастер – «Что вы знаете об этом объекте?» В ответ мы должны указать любое свойство или отношения, которое нам известно. В нашем примере мы знаем, что в этой стране находится некий курорт. Значит, в соответствии с нашей онтологией – существует объекты типа «Курорт», ссылающиеся на искомую страну. Выбираем соответствующий вариант в выпадающем меню мастера. Теперь искомым объектом становится «Курорт», и вопрос повторяется. О курорте нам известно, что он предлагает определенный вид отдыха. Искомым объектом становится «Вид отдыха». Об этом виде отдыха нам известна конкретная информация – значение свойства-литерала «Имя», равное «дайвинг». Выбираем соответствующий вариант в выпадающем меню, и появляется поле ввода значения свойства. В процессе ввода система подсказывает нам возможные варианты. В интерфейсе системы все это выглядит таким образом:

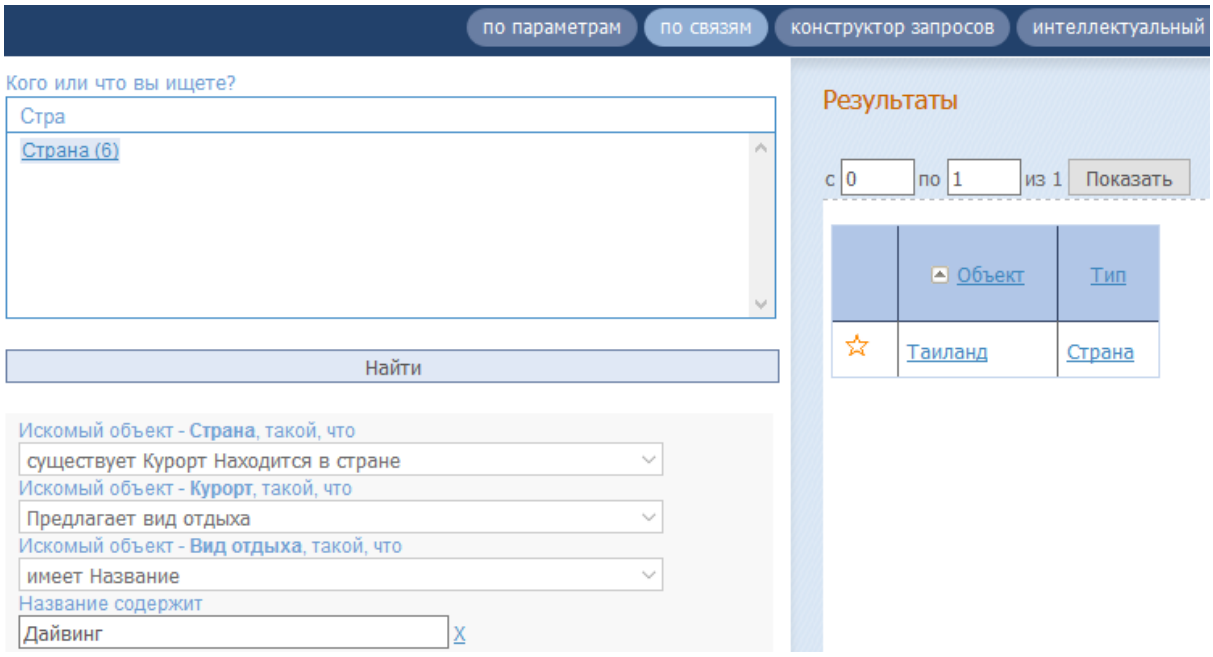

Рис. 50. Режим поиска по связям

При помощи такого интерфейса можно строить довольно сложные логические цепочки, но его ограниченность очевидна – например, мы не можем задать дополнительное условие (например, если нас интересуют только курорты, находящиеся в зоне тропического климата). Этот интерфейс позволяет пройти только один конкретный путь по графу, без ветвлений.

Лишен такого недостатка более сложный интерфейс – Конструктор запросов. Он построен на принципе создания системы логических уравнений с несколькими неизвестными. Фактически, он сводится к формулированию паттернов поиска триплетов, также, как и SPARQL-запрос, но, в отличие от последнего, не требует от пользователя познаний в программировании. Итак, пусть нас интересуют курорты, на которых можно заняться дайвингом или рафтингом, имеющие тропический климат.

В нашей системе уравнений будут две неизвестных величины – это курорт (обозначим его X), и вид отдыха (Y). Присутствует также одно известное значение – тропический климат. Зададим первое уравнение системы, которое выберет только X, имеющие тропический климат, и второе, которое свяжет X и Y: «X предлагает вид отдыха Y».

После этого нужно создать два условия на Y, сгруппированные между собой логическим оператором ИЛИ: Y имеет название Дайвинг, или Y имеет название Рафтинг. Для этого в форме конструктора предусмотрены возможности группировки условий (эквивалент «скобок» в уравнении), и объединения условий между собой логическими операторами И, ИЛИ, И НЕ.

Построенная система уравнений будет выглядеть в интерфейсе так:

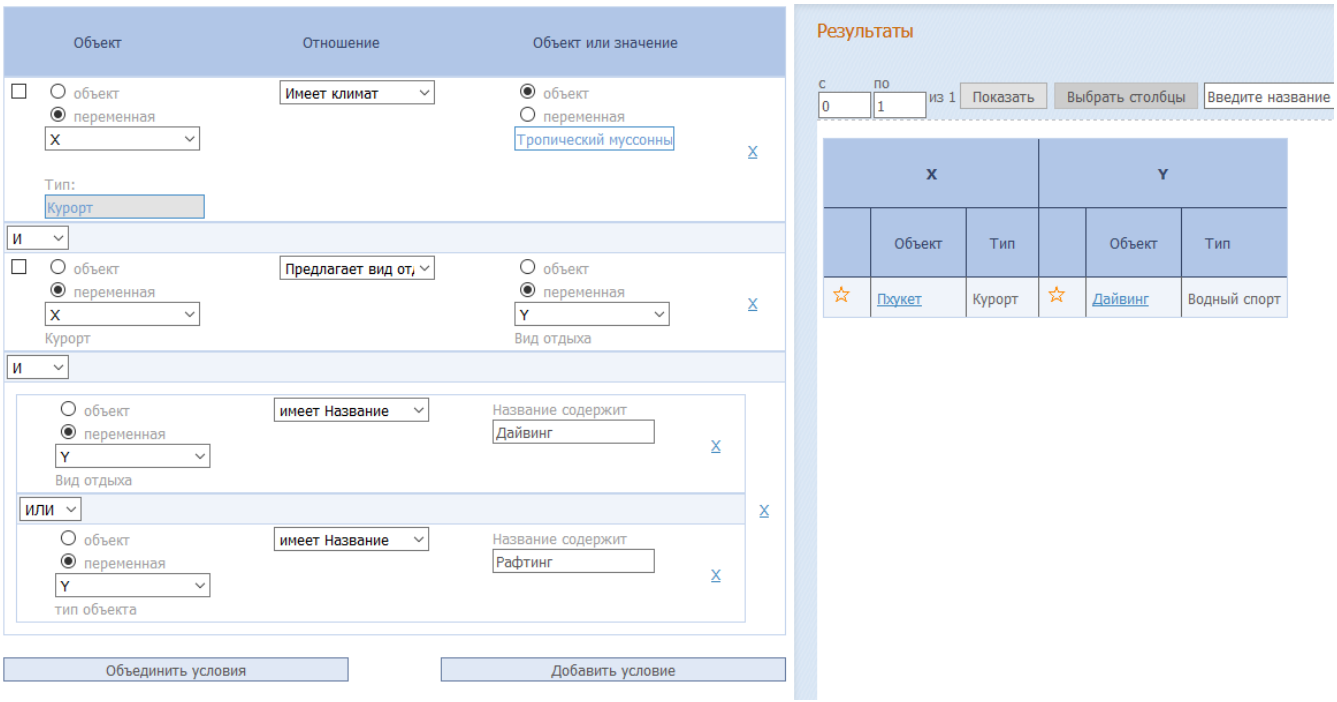

Рис. 51. Режим конструктора запросов

Нажатие кнопки «Найти» выдает нам множество решений – все возможные комбинации X и Y.

В процессе выбора/ввода значений выдаются подсказки, в том числе о том, какой тип имеют возможные значения каждой переменной. Этот интерфейс позволяет строить запросы любого уровня сложности, и находить решения любых задач, которые в принципе могут быть решены на данной онтологии.

Разумеется, возможности использования интерфейса не ограничиваются одномоментным получением результатов. Составив логическое уравнение, можно периодически анализировать изменения его результатов, и на этом основании формировать отчеты, уведомления, совершать любые другие программируемые действия.

Следующий режим, «интеллектуальный поиск», представляет собой конструктор запросов на контролируемом естественном языке. В процессе ввода поискового выражения система выдает подсказки, в соответствии с которыми

выбирается следующий термин. Например, только что описанный запрос будет выглядеть в этом режиме так:

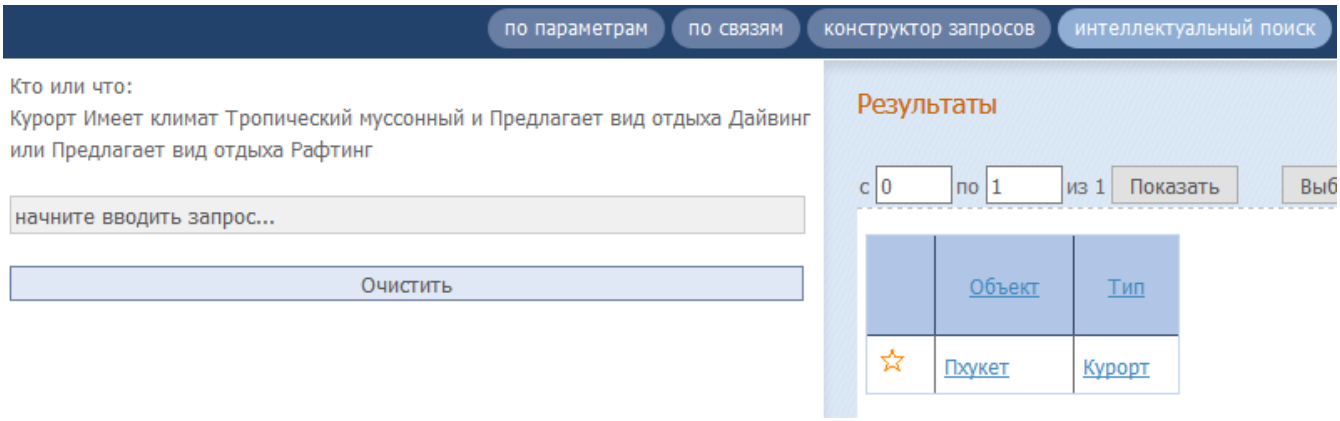

Рис. 52. Конструктор запросов на контролируемом естественном языке

Щелчок на любой найденный объект открывает страницу информации о нем, похожую по смыслу на вики-представление сведений об объектах в редакторе Onto.pro, но более широкую по содержанию. Страница имеет следующий вид:

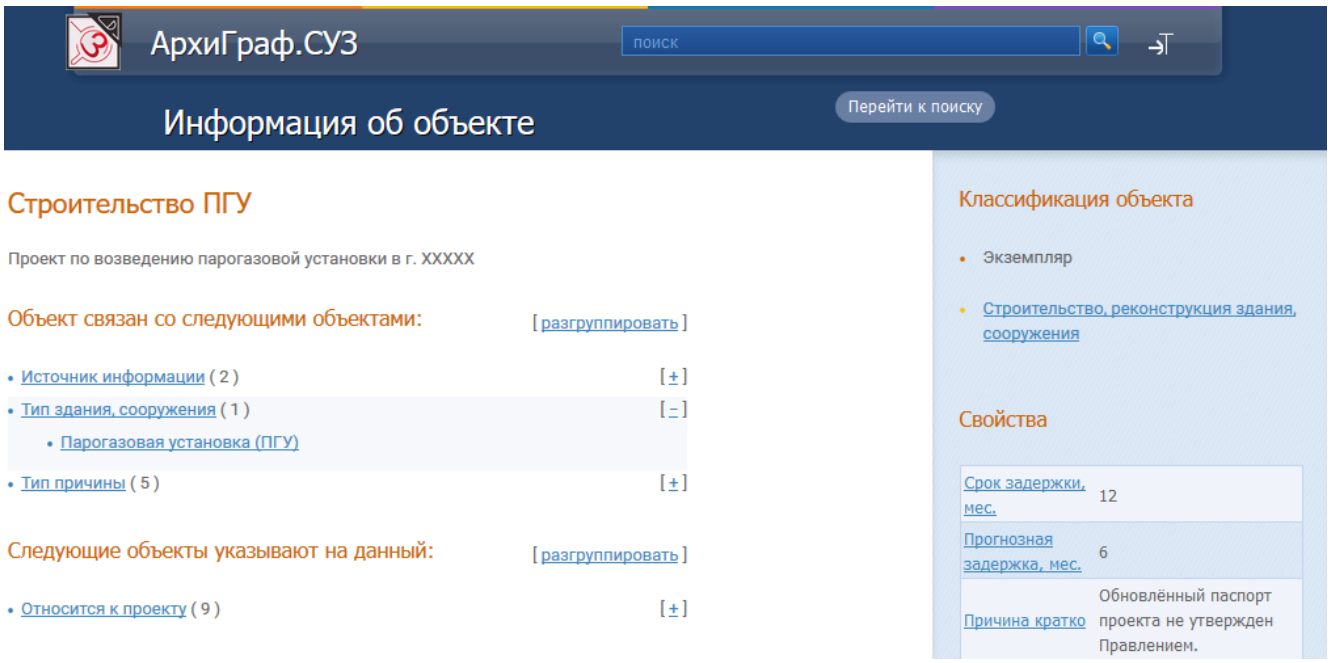

Рис. 53. Страница информации об объекте

В правой колонке отображается список классов, к которым относится объект. Ниже отображаются статические свойства данного объекта. В левой части страницы показаны связи, соединяющие этот объект с другими. Щелчком по любой

ссылке на этой странице можно перейти на такое же представление информации для соответствующего объекта, определения класса или свойства.

Внизу страницы отображается схема связей объекта в виде графа:

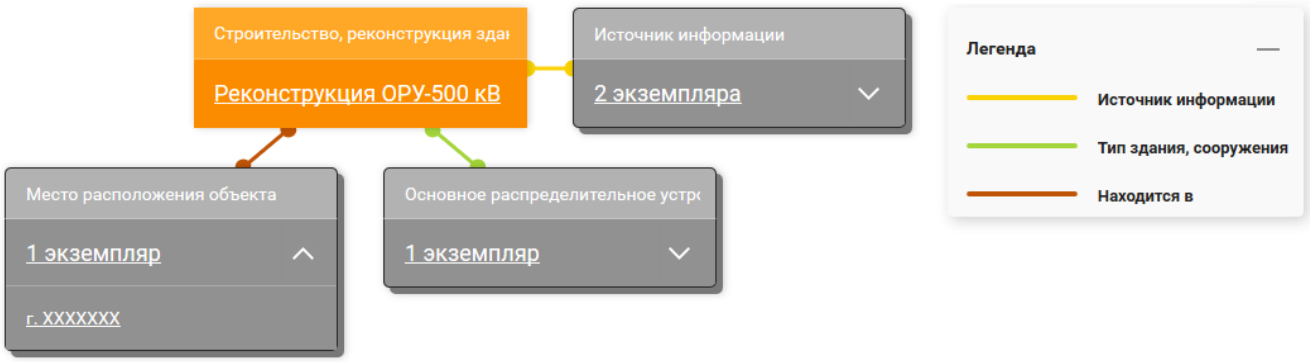

Рис. 54. Связи выбранного объекта

Текущий объект выделен цветом. Рядом в виде прямоугольников показаны все классы объектов, с которыми может быть связан текущий объект. Цвет соединяющей линии зависит от типа связи. Расшифровку цветов можно увидеть, раскрыв легенду в правой части изображения. Для каждого класса объектов показано число его экземпляров, связанных с текущим объектом. Нажатием на кнопку «вниз» можно раскрыть выпадающую часть блока, и увидеть список конкретных объектов.

Возможности системы АрхиГраф.СУЗ не исчерпываются интерфейсами поиска и просмотра информации. Так, она предлагает режим конструирования правил логического вывода, аналогичный по смыслу созданию SWRL-правил, но не требующий от пользователя навыков программирования.

Окно конструктора правил состоит из трех колонок:

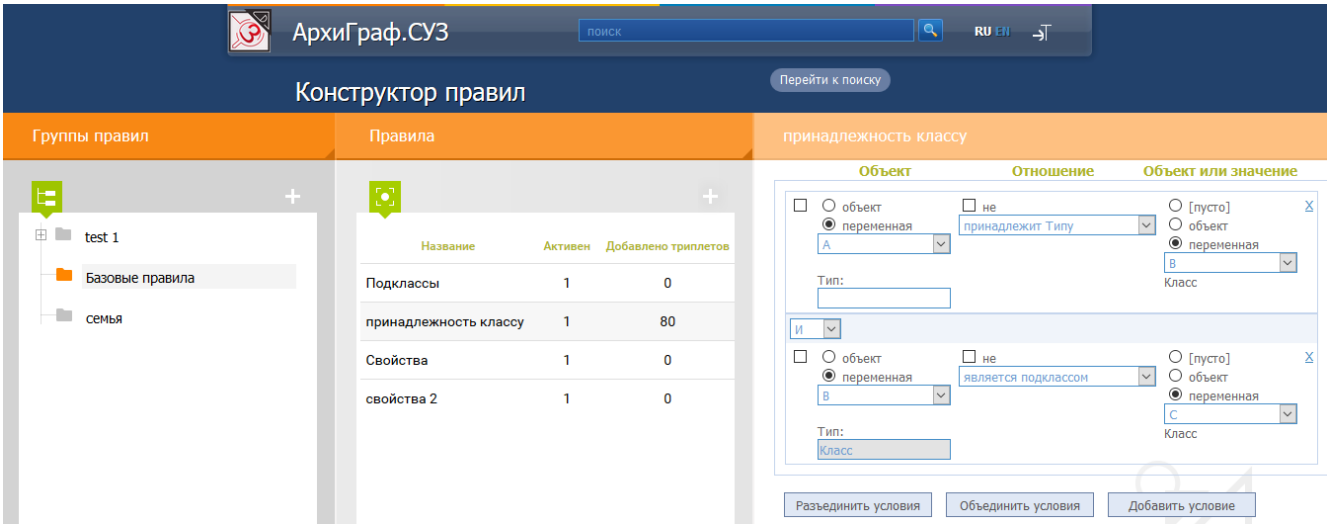

Рис. 55. Окно конструктора правил

Слева расположено дерево групп, содержащих правила. Правила выбранной группы показано в средней колонке показаны правила. Справа отображается информацию по выбранному правилу. Ниже в правой колонке находится форма настройки свойств правила.

Конструирование условий и вывода правила выполняется при помощи интерфейса, похожего на показанный выше на рис. 51 Конструктор запросов. АрхиГраф.СУЗ использует собственную машину логического вывода для вычисления результатов применения правил. Благодаря этому правила имеют более широкий в некоторых аспектах функционал, по сравнению с правилами SWRL. Так, в правилах АрхиГраф.СУЗ можно использовать отрицание, вывод может подразумевать создание нового объекта или удаление триплетов, имеющихся в базе. Как и Onto.pro, АрхиГраф.СУЗ работает с точкой доступа SPARQL напрямую, поэтому все результаты сохранения правил записываются в хранилище триплетов. При этом факты, заданные в онтологии в явном виде и полученные в результате логического вывода, сохраняются в разных именованных графах. При просмотре информации об объекте в СУЗ выведенные факты показываются другим цветом и содержат ссылку на правило, при помощи которого получены. В АрхиГраф.СУЗ реализована также возможность отслеживания результатов применения правил.

6. Семантические технологии: погружаемся в детали

Мы провели обзор методик и технологий создания и использования онтологических моделей. Теперь, сформировав представление о диапазоне этих средств, мы можем пойти на второй круг знакомства с ними, и более подробно рассмотреть некоторые важные детали. На этот раз мы последуем в обратном порядке: начнем с технологий, а в следующем разделе рассмотрим некоторые специальные вопросы методики онтологического моделирования.

6.1. Обзор возможностей OWL 1 и OWL 2

Ранее мы весьма поверхностно рассмотрели основные концепции RDF/RDFS/OWL, связанные с описанием классов, свойств, связей и индивидуальных объектов. Настало время узнать более подробно, какие технические средства моделирования предлагает стандарт OWL, но для этого нужно сначала рассмотреть одну возможность RDF, о которой мы умолчали – возможность создавать группировки объектов, не являющиеся классами.

Итак, RDF позволяет объявлять контейнеры трех типов:

1. rdf:Bag, объединяющий элементы модели или литералы без указания порядка их следования в группе.
2. rdf:Seq, делающий то же самое с указанием порядка следования элементов.
3. rdf:Alt, представляющий группу элементов модели или литералов, которые являются в определенном смысле альтернативами – например, для определенного значения некоторого свойства. Например, этот способ можно использовать для указания перечня страниц Интернета, где можно получить какую-либо информацию. При использовании такого контейнера приложение может предложить пользователю выбрать любой из содержащихся в нем вариантов.

Все эти способы применяются, как правило, для указания значений каких-либо свойств. Например, если в группу входят несколько студентов, значением

свойства «имеет студента» для группы может быть один из перечисленных контейнеров, объединяющий конкретных студентов.

Все перечисленные способы группировки не утверждают, что в создаваемых группах содержатся только перечисленные члены – иными словами, эти группы «открыты». Для создания «закрытых» групп, в которых участвует четко определенное число элементов, используются коллекции RDF. Для указания того, что определенный контейнер является коллекцией, используется атрибут rdf:parseType="Collection". Не будем останавливаться на конкретном синтаксисе, который можно найти в стандарте и многочисленных примерах к нему – для нас упоминание о коллекциях важно потому, что они потребуются при рассказе о возможностях собственно OWL.

Прежде всего отметим, что OWL определяет два предустановленных класса – owl:Thing и owl:Nothing. Первый из них объединяет все индивидуальные объекты модели, а его подклассами являются все ее классы. Второй не имеет экземпляров, и является подклассом любого класса онтологии.

Начнем обзор с рассмотрения способов объявления классов. Мы говорили, что класс – это именованная (или анонимная) группа, объединяющая индивидуальные объекты или другие классы. До сих пор мы использовали только один способ их объявления, и описали только один вид отношений между классами – отношение надкласс-подкласс.

Важным типом отношений между классами является указание их разделенности (предикат owl:disjointWith). Если задано такое отношение, то ни один индивидуальный объект, входящий в один из этих классов, не может одновременно входить и во второй – это будет означать неконсистентность онтологии.

Другой тип отношений между классами выражает их эквивалентность при помощи предиката owl:equivalentClass. Это выражение позволяет констатировать, что два класса с разными идентификаторами имеют в точности совпадающие наборы индивидуальных объектов.

Для объявления классов OWL предлагает целых шесть способов.

1. Простое объявление класса по идентификатору без указания дополнительных особенностей – это уже знакомый нам способ явного объявления именованного класса.
2. Перечисление. В этом случае класс задается явным перечислением входящих в него индивидуальных объектов. В синтаксисе RDF/XML такое определение может выглядеть следующим образом:

```
<owl:Class rdf:ID="#TerrestrialPlanet">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#Earth"/>
    <owl:Thing rdf:about="#Mars"/>
    <owl:Thing rdf:about="#Venus"/>
    <owl:Thing rdf:about="#Mercury"/>
  </owl:oneOf>
</owl:Class>
```

При таком объявлении (а также при всех последующих) класс может быть и анонимным, т.е. может иметь идентификатор, а может и не иметь.

Важная особенность классов, определенных при помощи перечисления, состоит в их «закрытости»: постулируется, что в класс входят эти и только эти индивидуальные объекты. Это открывает дорогу к получению логических выводов при помощи стандартных reasoner’ов, использующих парадигму открытого мира, которые не могли бы быть получены в ином случае. Приведем пример. Пусть класс «Член Совбеза ООН» делится на два разделенных подкласса: постоянных и не постоянных членов. Класс постоянных членов включает пять конкретных государств: РФ, США, Китай, Францию и Великобританию. Если мы определим класс не постоянных членов как дополнение к классу «Член Совбеза ООН» (об этом см. далее), то указание о том, что, например, Индия является членом Совбеза автоматически классифицирует ее как не постоянного члена. Это произойдет благодаря тому, что список постоянных членов определен

исчерпывающим образом. Если бы этот подкласс был определен любым другим способом, кроме перечисления, вывод, касающийся Индии, не был бы сделан – потому что в парадигме «открытого мира» предполагалось бы, что Индия может быть и постоянным членом Совбеза, просто этот факт не указан в онтологии.

3. Ограничение на свойства. Объявляет анонимный класс, который состоит из всех индивидуальных объектов, удовлетворяющих определенному условию. Условия могут быть двух типов. Первое из них формулируется как «все индивидуальные объекты, значение некого свойства для которых является объектом определенного типа». Например, так можно выделить класс объектов, которые ссылаются каким-либо определенным свойством на объекты класса «Процесс», с целью выделить в модели все сущности, связанные с процессами. В RDF/XML условие может выглядеть так:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#usedInProcess" />
  <owl:allValuesFrom rdf:resource="#Process"  />
</owl:Restriction>
```

Условие owl:allValuesFrom утверждает, что все значения свойства usedInProcess подходящих объектов должны указывать на объекты класса Process. Нюанс здесь состоит в том, что OWL использует концепцию «открытого мира», в соответствии с которой считается, что имеющаяся у нас онтология описывает не всю информацию – чего-то мы можем и не знать. Это напрямую влияет на применение правил логического вывода. Например, то, что все значения свойства usedInProcess определенного объекта относятся к классу Process, само по себе не сделает его членом того класса, для которого описано ограничение: считается, что могут существовать и другие значения свойства usedInProcess, о которых в нашей модели просто ничего не известно. Для того, чтобы вывод был получен, утверждение нужно «закрыть». Для этого нужно описать ограничение на количество значений, которые может принимать свойство usedInProcess.

Если будет известно, что оно принимает не более одного значения, это значение установлено, и указывает на объект класса Process – тогда вывод состоится.

Похожим образом описывается условие someValuesFrom, которое утверждает, что хотя бы одно из значений указанного свойства должно попадать в установленный диапазон. Например, таким образом можно описать тот факт, что членами класса «Работник», являющегося подклассом класса «Персоны», являются все объекты, которые имеют значение свойства «работает в», равное какому-либо объекту класса «Организация»:

```
<owl:Class rdf:about="#Employee">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#worksIn"/>
            <owl:someValuesFrom rdf:resource="#Organization"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>
```

Еще один вариант условия, owl:hasValue, утверждает, что значение свойства должно соответствовать конкретному индивидуальному объекту. Например, так можно выделить в класс все дочерние предприятия определенного холдинга:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasOwnerOrganization" />
  <owl:hasValue rdf:resource="#AAAHolding" />
</owl:Restriction>
```

Второй тип ограничений налагается на количество значений, которые может принимать свойство. Например, футбольная команда состоит из 11 игроков; следовательно, можно объявить класс, состоящий из объектов, у которых свойство «имеет игрока» принимает 11 значений – это будет класс футбольных команд. Можно описывать условия на минимальное,

максимальное и точное количество значений при помощи терминов owl:minCardinality, owl:maxCardinality и owl:cardinality:

```
<owl:Restriction>
  <owl:onProperty rdf:resource="#hasParent" />
  <owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">2</owl:maxCardinality>
</owl:Restriction>
```

С этим связан очень важный момент: описание ограничений в онтологии не налагает на самом деле никаких ограничений на то, какие значения могут принимать те или иные атрибуты. Вместо этого, ограничения утверждают, что все объекты, для которых они выполняются, относятся к определенному классу. Например, в случае с указанным выше ограничением, ничто не помешает нам присвоить трех родителей одному объекту. Однако, если мы создадим объект «Дружок», родителем которого объявим объект «Бобик», то с удивлением обнаружим в результатах логического вывода, что «Дружок» является человеком – ведь он имеет не более двух родителей, а ограничение мы налагали именно на класс «Человек».

Оказывается, ограничения – это никакие не ограничения, а правила отнесения объектов к классам! Впрочем, если вдуматься – это и является ограничением. Только на практике работа с ним не слишком удобна и очевидна.

4. Пересечение двух и более классов. Под «пересечением» понимаются объекты, входящие во все перечисленные классы одновременно (логическое «И»). Например, пересечением классов «Музей» и «Дворец» будут дворцы, эксплуатируемые как музеи – Эрмитаж, Лувр и т.д. Третьяковская галерея и Большой Кремлевский дворец в это пересечение не попадут. В RDF/XML пересечение объявляется так:

```
<owl:Class>
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Palace"/>
    <owl:Class rdf:about="#Museum"/>
  </owl:intersectionOf>
</owl:Class>
```

5. Объединение двух и более классов. Операция объединения соответствует логическому «ИЛИ»: ее результатом будут все индивидуальные объекты, входящие хотя бы в один из объединяемых классов. Так, можно объявить класс домашних животных, как объединение классов кошек и собак:

```
<owl:Class>
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Cat"/>
    <owl:Class rdf:about="#Dog"/>
  </owl:unionOf>
</owl:Class>
```

6. Дополнение описания класса. Эта операция соответствует логическому «НЕ»: ее результатом являются все индивидуальные объекты, не относящиеся к указанному классу. Например, мы можем объявить класс «Объекты» как все, что не относится к «Субъектам»:

```
<owl:Class>
  <owl:complementOf>
    <owl:Class rdf:about="#Subject"/>
  </owl:complementOf>
</owl:Class>
```

Состав членов классов, объявленных способами, описанными в п.п. 4-6, может определить только машина логического вывода.

Перейдем к рассказу о свойствах. Как мы помним, свойства в OWL бывают двух основных типов: свойства-литералы (owl:DatatypeProperty) и свойства-связи (owl:ObjectProperty). И те, и другие являются подклассом свойства вообще: rdf:Property. Кроме них, существуют и дополнительные, служебные виды свойств, такие как owl:AnnotationProperty.

Свойства могут образовывать иерархии точно так же, как классы, при помощи аксиомы rdfs:subPropertyOf. Например, объявление

```
<owl:ObjectProperty rdf:ID="isMother">
  <rdfs:subPropertyOf rdf:resource="#isParent"/>
</owl:ObjectProperty>
```

означает, что свойство «является матерью» – специализация свойства «является родителем». Следовательно, по правилам логического вывода, если A является матерью B, то A является и родителем B, но не наоборот.

Отношение owl:inverseOf связывает между собой инверсные свойства, о которых мы уже говорили ранее. Еще одно описанное выше отношение, owl:propertyDisjointWith, заданное для свойств C и D, утверждает, что если индивид A связан с B свойством C, то A НЕ связан с B свойством D.

Интересно, что в OWL DL свойства-литералы и свойства-связи разобщены, а в OWL Full свойства-литералы являются подтипом свойств-связей.

Уже знакомые нам предикаты rdfs:domain и rdfs:range используются для указания классов, к объектам которых применимо данное свойство, и диапазонов значений свойств. Как мы уже говорили, указание нескольких domain и range для свойства интерпретируется так, будто между ними стоит операция логического «И», то есть (в случае domain) свойством могут обладать только объекты, относящиеся одновременно ко всем перечисленным классам – эта особенность стандарта тоже не слишком интуитивна. Если требуется выразить тот факт, что свойством может обладать объединение подклассов – необходимо использовать owl:unionOf:

```
<owl:ObjectProperty rdf:ID="hasBankAccount">
  <rdfs:domain>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Person"/>
        <owl:Class rdf:about="#Organization"/>
      </owl:unionOf>
    </owl:Class>
  </rdfs:domain>
</owl:ObjectProperty>
```

В данном случае утверждается, что владельцем банковского счета могут быть как персоны, так и организации.

Много сведений о свойствах можно выразить путем их отнесения к определенным в стандарте специальным классам (типам). Так, отнесение свойства к типу owl:FunctionalProperty является «глобальным ограничением»: оно констатирует, что каждый носитель этого свойства может иметь одно и только одно его значение. Owl:FunctionalProperty является подклассом rdf:Property. Для того, чтобы объявить свойство функциональным, нужно отнести его к этому типу, наряду с основным имеющимся у него типом – ObjectProperty или DatatypeProperty:

```
<owl:ObjectProperty rdf:ID="husband">
  <rdf:type    rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Woman" />
  <rdfs:range  rdf:resource="#Man" />
</owl:ObjectProperty>
```

В этом примере утверждается, что у каждой женщины может быть только один муж.

Могут существовать и обратные функциональные свойства, owl:InverseFunctionalProperty – в этом случае на каждый объект может указывать только одна связь данного типа. Например, выражение

```
<owl:InverseFunctionalProperty rdf:ID="biologicalMotherOf">
  <rdfs:domain rdf:resource="#Woman"/>
  <rdfs:range rdf:resource="#Human"/>
</owl:InverseFunctionalProperty>
```

констатирует, что у каждого человека есть только одна биологическая мать, при этом свойство «является биологической матерью» указывает от женщины на человека.

Свойства могут находиться в определенных отношениях между собой. Выражение owl:equivalentProperties констатирует, что два свойства, обладающие разными идентификаторами, имеют один и тот же смысл. Выражение owl:inverseOf указывает, что одно свойство является обратным другому:

```
<owl:ObjectProperty rdf:ID="hasChild">
  <owl:inverseOf rdf:resource="#hasParent"/>
</owl:ObjectProperty>
```

Очевидно, что если A является родителем B, то B является ребенком A. Такие утверждения образуют одно из правил, учитываемых машинами логического вывода.

Наконец, свойства сами по себе могут обладать определенными логическими характеристиками. Свойство может быть транзитивным, owl:TransitiveProperty – это означает, что если A связано неким свойством с B, а B с C, то A связано этим же свойством с C. Такое объявление полезно для автоматического вычисления на описаниях вложенных структур. Например, описывая состав какого-либо изделия, мы будем использовать свойство «является частью» для его декомпозиции до нужного нам уровня. Например, турбину электростанции можно рекурсивно декомпозировать на узлы, компоненты и т.д., однако очевидно, что самый последний болт является частью турбины в целом, а не только ее конкретного узла. Для этого и пригодится определение транзитивного свойства:

```
<owl:ObjectProperty rdf:ID="isPartOf">
  <rdf:type rdf:resource="&owl;TransitiveProperty"/>
  <rdfs:domain rdf:resource="#Asset"/>
  <rdfs:range  rdf:resource="#Asset"/>
</owl:ObjectProperty>
```

Впрочем, owl:TransitiveProperty является подтипом owl:ObjectProperty, поэтому объявлять ObjectProperty в явном виде в данном случае не обязательно.

Свойство может быть симметричным: это означает, что если A имеет некую связь с B, то и B имеет такую же связь с A. Например, для отношения «является супругом» запись будет такой:

```
<owl:SymmetricProperty rdf:ID="hasSpouse">
  <rdfs:domain rdf:resource="#Human"/>
  <rdfs:range  rdf:resource="#Human"/>
</owl:SymmetricProperty>
```

Еще один тип свойств, owl:ReflexiveProperty, утверждает, что это свойство связывает каждый объект сам с собой. В качестве примера можно привести свойство «совпадает по размеру», т.к. каждый объект совпадает по размеру сам с собой.

Рассмотрим особенности OWL, связанные с описанием индивидуальных объектов. Прежде всего отметим, что индивидуальные объекты, как и классы, могут быть анонимными, то есть не обладать уникальными идентификаторами. Это не слишком удобно при работе с ними через SPARQL-запросы, так как подобные сущности представляются пустыми узлами, blank nodes; однако, стандарт допускает такой способ работы.

Другой важный момент связан с тем, что один и тот же объект может иметь несколько разных идентификаторов в модели. По умолчанию считается, что у машины логического вывода нет информации о том, чтобы считать любые два объекта модели эквивалентными или различными. Для явного указания таких фактов используются выражения owl:sameAs, owl:differentFrom и owl:AllDifferent. Как понятно из названия, owl:sameAs констатирует эквивалентность объектов:

```
<rdf:Description rdf:about="#ЛевНиколаевичТолстой">
  <owl:sameAs rdf:resource="#ЛевТолстой"/>
</rdf:Description>
```

Выражение owl:differentFrom аналогичным образом декларирует уникальность объектов. Наконец, owl:AllDifferent является специальным классом OWL, включение объектов в который декларирует тот факт, что все они являются уникальными, чтобы избежать множества парных разобщений. Пример может выглядеть так:

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Person rdf:about="#KarlMarx"/>
    <Person rdf:about="#FriedrichEngels"/>
  </owl:distinctMembers>
</owl:AllDifferent>
```

Оба эти способа не слишком удобны, так как утверждения об уникальности индивидуальных объектов придется дополнять после создания каждой новой сущности. Избежать этого позволяет объявление какого-либо свойства как ключевого для объектов того или иного класса при помощи конструкции hasKey, доступной в стандарте OWL 2. В этом случае все объекты, обладающие одинаковым

значением этого свойства, будут считаться эквивалентными, а при различии значений – уникальными.

Все описанное нами в этой главе (кроме конструкции HasKey) относится к стандарту OWL 1. Стандарт OWL 2, принятый в 2012 году, расширяет его возможности.

Мы уже упоминали о том, что он определяет три «профиля», каждый из которых налагает определенные ограничения на возможности языка, с целью достижения приемлемой производительности в тех или иных практических ситуациях. Перечислим профили OWL 2:

1. OWL 2 EL оптимизирован для обработки онтологий с большим числом классов и свойств.
2. OWL 2 QL больше подходит для онтологий с большим количеством индивидуальных объектов.
3. OWL 2 RL применяется в случаях, когда нужно добиться приемлемой производительности без существенной потери выразительности языка.

Перечислим некоторые наиболее интересные функциональные возможности OWL 2.

- Возможность описывать цепочки свойств, что позволяет, например, описать отношения между тремя объектами сразу, которой так не хватало в OWL 1. Для этого используется выражение ObjectPropertyChain вместе с SubObjectPropertyOf. Например, если ObjectPropertyChain связывает свойства «находится в» и «является частью», и эта цепочка является SubObjectPropertyOf для свойства «находится в», то турбина, являющаяся частью агрегата, находящего в цеху №1, будет считаться тоже находящейся в цеху №1. Еще один

> *Невозможность описать правило установления отношения «является дядей» была хрестоматийным примером и главным упреком стандарту OWL 1 со стороны скептиков.*

классический пример использования цепочек свойств – отношение «является дядей», которое устанавливается при наличии двух отношений между тремя субъектами: A имеет родителя B, B имеет брата C. Правда, такая реализация сразу порождает методическую проблему: если B умрет и перестанет быть членом онтологии, C не перестанет быть дядей A, хотя reasoner сделает именно такой вывод. У этой проблемы есть несколько решений, простейшее из которых – не удалять из онтологии индивиды, описывающие ранее существовавшие объекты реального мира, а помечать их значением определенного свойства. Это приводит нас к вопросу о том, как вообще отражать в онтологиях время и эволюцию систем – ведь до сих пор мы описывали только текущее, мгновенное их состояние. О некоторых подходах к созданию онтологических моделей динамических систем мы расскажем далее.

- Возможность объявлять определенные наборы свойств ключевыми (HasKey) – используется для того, чтобы констатировать, что каждый экземпляр какого-либо класса исчерпывающе идентифицируется набором значений своих ключевых свойств.
- Явное описание не рефлексивных, не симметричных, разделенных свойств (IrreflexiveObjectProperty, AsymmetricObjectProperty, DisjointObjectProperties).
- Отрицательные утверждения (например, при помощи выражения NegativeObjectPropertyAssertion можно записать аксиому «Россия не расположена в Африке»).

Рассказав о возможностях OWL, мы обязаны упомянуть о его ограничениях.

Одним из важнейших из них считается невозможность выполнять вычисления на основе нечеткой логики. Действительно, мы можем напрямую записать в модели факт «В Петербурге *часто* идет дождь», но это не позволит автоматически в одних случаях получать вывод о том, что дождь идет, а в других – нет. Однако на уровне

ПО, работающего с моделью, никто не мешает сопоставить предикату «часто» конкретную функцию, которая будет выдавать некое распределение, на основании которого система имитационного моделирования будет попеременно выдавать разные варианты погоды в Петербурге. Кроме того, можно задать более формальное описание в самой модели, выразив сведения о том, что вероятность дождя в такие-то периоды имеет определённое значение. Пример подобного использования модели мы продемонстрируем в главе 7.

Похожее рассуждение относится к модальной логике, которая позволяет не только описывать фактическое состояние систем, но и выражать суждения о том, что может, должно, могло бы быть.

Аналогичным образом, недостатком OWL иногда называют невозможность выражать сведения о субъективной точке зрения: «Иван считает, что пингвины живут в Арктике». Иван, конечно, ошибается, но инструментальных средств для выражения этого в OWL, казалось бы, нет. Однако никто не мешает в модели создать класс объектов «Мнение», который будет связывать субъекта с теми фактами, которые он считает истинными.

Наконец, проблемой может показаться описание исключений. Факт «птицы летают» на первый взгляд кажется истинным, однако куры и пингвины не летают. Это говорит только о том, что утверждение сформулировано неправильно, и его нужно уточнить. Летают не птицы, а только определенные их виды – это и нужно отразить в модели. Либо следует добавить модальность, сообщив, что *почти* все птицы летают – тогда выводы, полученные на основании этого правила, будут верны с определенной долей достоверности.

Действительно трудноразрешимой проблемой для OWL является представление утверждений о классах. Если «Собака» – класс, то выразить напрямую сведения о том, что все собаки умеют лаять, будет сложно. Для этого потребуется создать класс «Умение», один из объектов которого будет соединять класс Собаки и умение лаять. ПО, использующее модель, сможет интерпретировать эту информацию, но задействовать её в получении логических выводов средствами

reasoner'a не получится. Другой вариант решения проблемы – создание правила, которое гласит, что если X – собака, то X умеет лаять. Это сработает, но большими массивами правил трудно управлять, особенно если онтология подвергается рефакторингу.

В целом, выразительные возможности OWL, как и любого другого инструмента моделирования, не исчерпывающи. Однако при современном состоянии дел в области создания и использования моделей следует признать, что полное раскрытие и внедрение в практику даже этих возможностей даст такой уровень качества практических результатов, о котором без этого не приходится и мечтать. Затем, когда недостатки и ограничения OWL станут реальным препятствием для дальнейшего повышения качества результатов, резонно будет поставить вопрос о выборе или разработке технологий следующего уровня.

6.2. Обзор возможностей SPARQL. Entailment режимы

Мы уже не раз упоминали о том, что при определенных обстоятельствах в хранилище триплетов образуются так называемые «пустые узлы» (blank nodes). Рассмотрим конкретный пример. Импортируем через SPARQL-интерфейс OWL-файл, содержащий один из примеров из предыдущей главы:

```
<owl:Class rdf:about="#Employee">
    <owl:equivalentClass>
        <owl:Restriction>
            <owl:onProperty rdf:resource="#worksIn"/>
            <owl:someValuesFrom rdf:resource="#Organization"/>
        </owl:Restriction>
    </owl:equivalentClass>
    <rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>
```

Этому будет соответствовать следующий набор триплетов:

<http://trinidata.ru/test/Employee>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#Class>

<http://trinidata.ru/test/Employee>	<http://www.w3.org/2000/01/rdf-schema#subClassOf>	<http://trinidata.ru/test/Person>
<http://trinidata.ru/test/Employee>	<http://www.w3.org/2002/07/owl#equivalentClass>	_:b0
_:b0	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#Restriction>
_:b0	<http://www.w3.org/2002/07/owl#someValuesFrom>	<http://trinidata.ru/test/Organization>
_:b0	<http://www.w3.org/2002/07/owl#onProperty>	<http://trinidata.ru/test/worksIn>

Выражения вида _:b0 – это временные идентификаторы пустых узлов, blank nodes. Такие узлы образуются при создании элементов, не имеющих собственных идентификаторов – в нашем случае это анонимный класс, объявленный выражением owl:Restriction. Печальная особенность пустых узлов состоит в том, что их идентификаторы могут отличаться в результатах разных запросов; эти идентификаторы нельзя использовать в запросах на удаление триплетов. Таким образом, удалить их можно только при помощи запросов DELETE WHERE с подходящим паттерном.

Разумеется, прямая работа с пустыми узлами через SPARQL-запросы не очень удобна, но они для этого и не предназначены. При помощи таких элементов задаются правила логического вывода, которые автоматически применяются хранилищем триплетов при работе в entailment-режимах. Мы уже упоминали о том, что в этих режимах (их несколько, в соответствии со стандартами и диалектами RDF/RDFS/OWL) точка доступа выполняет логический вывод по правилам, установленным в стандарте, и возвращает не только триплеты, непосредственно помещенные в модель, но и триплеты, полученные в результате логического вывода. Недостаток здесь состоит в том, что при работе через SPARQL точку доступа невозможно отличить «выведенные» триплеты от явно внесенных знаний.

В конфигурационном файле Fuseki режим OWL entailment включается так:

```
<#model_inf> a ja:InfModel ;
     ja:baseModel <#tdbGraph> ;
     ja:reasoner [
```

```
    ja:reasonerURL <http://jena.hpl.hp.com/2003/OWLFBRuleReasoner>
] .
```

Нужно отметить, что работа с entailment режимом приводит к существенному замедлению выполнения запросов. Алгоритм логического вывода состоит в том, что reasoner в цикле итеративно применяет все имеющиеся правила логического вывода до тех пор, пока на определенном шаге цикла не будет получено новых триплетов. Разумеется, при сложных зависимостях между сущностями, глубоких иерархиях классов такие вычисления могут быть очень длительными. В связи с этим, например, созданный нами редактор онтологий Onto.pro не использует entailment режим, а самостоятельно эмулирует применение многих правил, необходимых для решения его задач.

Работает entailment режим предельно просто. Например, если загрузить в точку доступа приведенный выше фрагмент OWL с описанием ограничения, а также сообщить факты о том, что персона Smith работает в организации Alpha, при выполнении запроса

```
SELECT * WHERE { test:Smith ?b ?c }
```

получим следующий набор триплетов:

<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#NamedIndividual>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://trinidata.ru/test/Person>
<http://trinidata.ru/test/worksIn>	<http://trinidata.ru/test/Alpha>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://trinidata.ru/test/Employee>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	_:b0
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2000/01/rdf-schema#Resource>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.w3.org/2002/07/owl#Thing>
<http://www.w3.org/2002/07/owl#sameAs>	<http://trinidata.ru/test/Smith>

«Выведенные» триплеты выделены серым. Как видно, среди них немало не очень осмысленных (три последних), но есть и содержательная информация – выводы о том, что Smith является членом класса Employee, и анонимного класса-ограничения, обозначенного пустым узлом _:b0.

Синтаксис SPARQL достаточно развит, и содержит немало «синтаксического сахара» – например, для работы с коллекциями RDF. Описание всех этих нюансов можно найти в стандарте и многочисленных статьях о нем. Здесь мы перечислим лишь несколько конкретных возможностей стандарта, наиболее часто применяемых на практике.

1. Группировка паттернов. Паттерны внутри выражения, заключенного в фигурные скобки, могут объединяться в подгруппы при помощи вложенных фигурных скобок. Эти подгруппы подходят для применения фильтров, которые действуют только внутри той подгруппы, в которой находятся.
2. Перед подгруппами может употребляться уже знакомое нам ключевое слово OPTIONAL. В этом случае будут найдены решения для основного паттерна, даже если решений для опциональной подгруппы не найдется:

```
{ ?person test:worksIn ?company
OPTIONAL { ?company rdfs:label ?name } }
```

   - компания может и не иметь заданного имени, но все отношения «A работает в B» все равно будут возвращены.

   Если условие употребляется в опциональной подгруппе, и оно не выполнено – то в ответ на запрос не попадет только опциональная подгруппа, а решения основного паттерна будут возвращены в любом случае.
3. Перед подгруппами может употребляться ключевое слово UNION, обозначающее логическое объединение («ИЛИ») решений двух подгрупп. Например:

```
{ { ?asset test:situatedIn ?place } UNION { ?asset test:belongsTo ?place } }
```

Этот паттерн вернет все активы, связанные с местами их нахождения как свойством situatedIn, так и belongsTo.

Кроме этого, в стандарте SPARQL определено немало других возможностей, в частности – использование различных операторов. Здесь мы рассмотрим еще два компонента SPARQL – работу с именованными графами, и федерированные запросы.

Хранилище триплетов можно разделить на сегменты, называемые именованными графами (named graph). Все SPARQL-запросы, которые мы рассматривали до сих пор, выполнялись на графе по умолчанию, который не имеет имени. Однако существует возможность создать именованный граф командой

```
CREATE GRAPH <http://trinidata.ru/testgraph>
```

и после этого использовать его в запросах. Например, для помещения данных в этот граф надо добавить в запрос ключевое слово GRAPH:

```
INSERT DATA { GRAPH <http://trinidata.ru/testgraph> { test:Object rdfs:label “name” } }
```

Аналогично работает это ключевое слово и в других запросах – DELETE, SELECT:

```
SELECT * WHERE { GRAPH <http://trinidata.ru/testgraph> { test:Object ?prop ?value }}
```

Если граф явно указан в запросе, то данные будут возвращены только из основного графа. Если нужно соединить в ответе триплеты из нескольких графов, вместо конкретного имени графа можно указать переменную:

```
SELECT * WHERE { GRAPH ?src { test:Object ?prop ?value } }
```

Стандарт предусматривает и другие способы указания графа в запросе – ключевые слова WITH, USING. Однако не все программные реализации одинаково их поддерживают. Существуют команды для копирования, переноса триплетов между графами (COPY, MOVE).

Fuseki, кроме того, предлагает возможность автоматически дублировать все триплеты из именованных графов в основной – для этого используется директива tdb:unionDefaultGraph true в конфигурационном файле. Проблема состоит в том, что

после включения этой опции точка доступа оказывается в read only режиме: запросы на изменение данных работать при этом не смогут.

Интересной является возможность выполнения так называемых федерированных (Federated) запросов, когда разные части графа находятся в разных хранилищах триплетов. Такие запросы используются, например, в случае, если одни онтологии (стандартные) используются как основа для создания других. Предположим, что в той точке доступа, к которой мы выполняем запрос, хранятся сведения о сотрудниках компаний, а имена этих компаний хранятся в другой точке. Тогда запрос на вывод списка сотрудников вместе с названиями компаний будет выглядеть так:

```
SELECT ?person_name ?company_name WHERE {
  ?person <http://example.com/#WorksIn> ?company .
  ?person rdfs:label ?person_name .
  SERVICE <http://11.22.33.44/sparql> {
    ?company rdfs:label ?company_name }
}
```

Часть паттерна, находящаяся в клаузе SERVICE, адресуется другой точке доступа. http://11.22.33.44/sparql - адрес HTTP-интерфейса этой точки (тот же самый адрес, который мы видим, заходя в панель управления точки доступа). Главным неудобством является необходимость знать адреса точек доступа на момент составления запроса, а также иметь точные сведения о том, в какой точке доступа размещена какая часть модели. Штатные средства каталогизации частей модели на сегодняшний день в SPARQL отсутствуют.

7. Методические вопросы онтологического моделирования

В этом разделе мы обсудим некоторые практические вопросы, возникающие в процессе концептуального моделирования, и воплощения созданных моделей при

помощи OWL. Мы не претендуем на изложение полной и исчерпывающей методики моделирования, подобной тем, что приводится в трех основополагающих книгах, упомянутых в начале нашего пособия. Наша цель – осветить некоторые теоретические и практические вопросы, которые позволят эффективно организовать мышление при создании моделей.

7.1. Знак и означаемое. «Правильность» модели.

Проблема знака и означаемого подробно изучается семиотикой, специальным разделом философии. Главным тезисом, который необходимо усвоить и постоянно применять при моделировании, является принципиальная их разделенность. Знак не тождественен означаемому, не имеет с ним никакой связи, кроме той, которую мы произвольно установили в своем сознании. Более того: именование всегда связано с упрощением, редукцией бесконечно разнообразного потока воспринимаемой информации о реальном мире к конечному информационному сигналу.

Процитируем слова из Алмазной сутры, одного из основополагающих текстов буддизма:

> О всех пылинках Так Приходящий проповедовал как о не-пылинках.
> Это и называют пылинками.
> Так Приходящий проповедовал о мирах как о не-мирах.
> Это и называют мирами.

Эти слова постулируют принципиальную разделенность, не тождественность знака и означаемого. Называя нечто «пылинкой», мы отбрасываем значительную часть информации об этом объекте (что уж говорить о том, что мы называем «миром»). Создаваемый в нашем сознании образ пылинки, конечно, не является самой пылинкой. Но именно такие образы мы и называем пылинками – а не сами бесконечно разнообразные, не познаваемые пылинки, существующие в реальном мире.

С этим вопросом напрямую связан другой – о качестве, «правильности» получаемых моделей. Слово «правильность» мы заключили в кавычки именно потому, что нет и не может быть однозначного теоретического критерия корректности модели. Единственной точкой отсчета, на которую можно опереться, является практическая пригодность полученных результатов моделирования для использования в деятельности субъекта, в соответствии с его целями. Такая формулировка говорит о том, что модель, «правильная» для одного субъекта, будет «неправильной» для другого. Из этого следует, что не существует никаких «предпочтительных» или «лучших» методик моделирования или концептуализации – существуют лишь методики, которые позволяют достичь цели с приемлемым уровнем затрат, и методики, которые не позволяют этого сделать.

Вопрос для размышления

Представьте, что мы моделируем дорожное движение в городе. Нужно ли учитывать в ней, что легковые автомобили имеют разный размер и мощность двигателей?

Оставаясь в заданной философской модальности, предположим, что некто пытается составить модель (или образ) песка, который несут воды Ганга. Ему может показаться, что если он представит каждую отдельную песчинку, то его модель будет правильной, верной. Но сколько бы миллиардов песчинок он не представил, его модель будет не лучше, чем та, в которой сказали одну фразу – "песок, который несут воды Ганга". Ибо и Ганг, и его воды, и песок в них, существуют только в наших глазах и в нашем сознании. Вещество-то, конечно, существует и без нас, но только мы выделяем в этом веществе части, и нарекаем их Гангом, водами и песком.

Однако если некто пытается не просто составить модель «песка, который несут воды», но и, например, спрогнозировать изменение гранулометрического и минерального состава аллювиальных отложений в дельте Ганга под действием антропогенных факторов – тогда, конечно песчинки в модели нужно будет учитывать. Правда, все равно не на уровне отдельных песчинок, а на уровне их

значимых типов. Такая постановка задачи наделит песчинки свойствами, заинтересуется их происхождением, скоростью оседания в потоке, и так далее. Желательно при этом помнить, что у песчинок нет гранулометрической классификации – она есть у образов песчинок, существующих для субъекта, решающего свою задачу.

Так каким же образом проверить, что созданная модель корректна? Сформулируем общий критерий проверки правильности модели процесса с прагматической точки зрения. Фрагмент реальности и его модель связаны отношениями подобия, через которые выражаются те правила, которые заложены в выбранной методике моделирования. Правила позволяют переходить от рассмотрения реального мира к модели и обратно. Методику выбирается исходя из приемлемого для наших задач уровня абстракции, по описанному выше принципу.

У моделируемого процесса есть исходное и конечное состояния. При помощи отношений подобия мы можем «отразить» их в модель. Так вот, если результат протекания процесса в реальном мире, отраженный в модель, и результат протекания модели процесса в самой модели, будут отличаться не более, чем на устраивающую нас величину погрешности — значит, модель пригодна. На диаграмме это можно показать так:

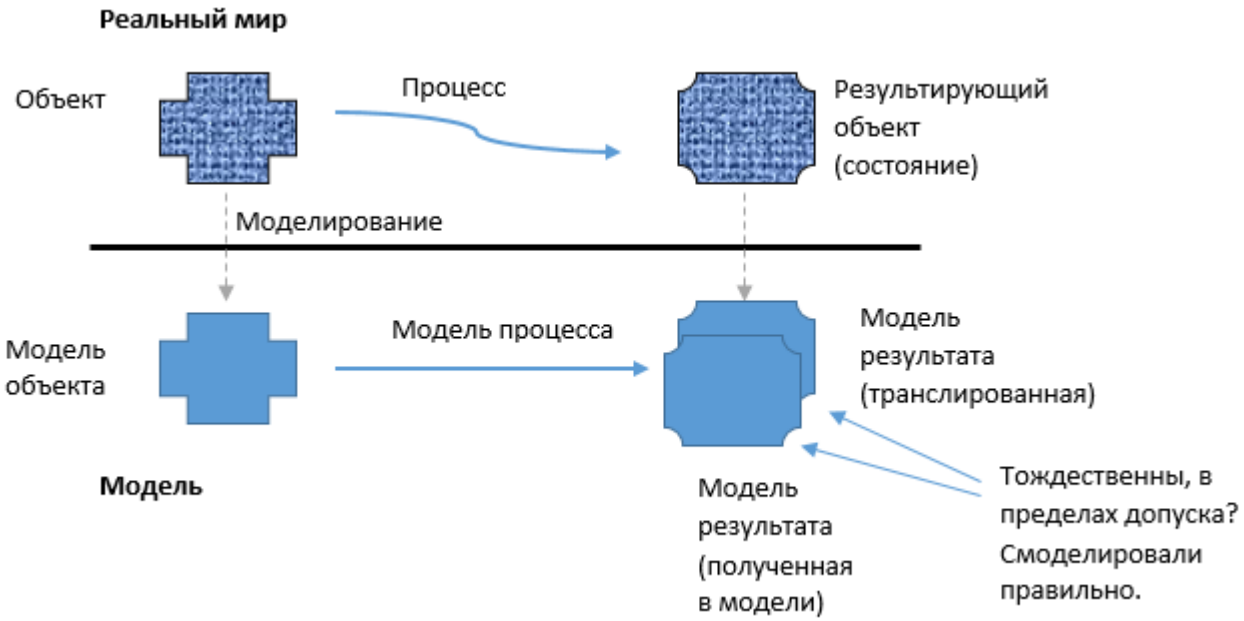

Рис. 56. Критерий корректности модели

Из этого следует, что модель может быть сколь угодно простой и примитивной. Это допустимо и даже хорошо, если она позволяет достичь результата.

7.2. Моделирование сложных систем

Во введении к нашему пособию мы уже говорили о том, что управление, основанное на результатах математического моделирования сложных систем, имеет серьезные недостатки. Рассмотрим причины возникновения этих недостатков, а затем расскажем о более продуктивных подходах к моделированию систем.

Итак, традиционный способ решения любых оптимизационных задач традиционно заключается в:

- Формализации фрагмента предметной области таким образом, который позволяет отразить ее характеристики, существенные для оценки результата;
- Выборе способа представления характеристик в виде чисел, установления зависимостей между характеристиками;
- Разделении характеристик на заданные в качестве ограничений, и подлежащие оптимизации;
- Поиске экстремальных значений оптимизируемых характеристик.

Как мы уже указывали, главной проблемой на этом пути является то, что ради достижения «понятности» модели и удобства выполнения вычислений делаются слишком серьезные упрощения, неправомерные предположения, избавиться от которых в дальнейшем оказывается невозможно. Может возникнуть ситуация, описанная в пословице: «гладко было на бумаге, да забыли про овраги». Сформированная в результате оптимизации программа действий оказывается невыполнимой из-за неучтенных в модели факторов или не предсказанных моделью побочных эффектов.

> *"Оптимизация параметров плана может приводить к полному уничтожению планируемой системы, вследствие возникающей из-за оптимизации неустойчивости."*
> *"Жесткую модель всегда надлежит исследовать на структурную устойчивость полученных при ее изучении результатов по отношению к малым изменениям модели."*
> *ак. В.И. Арнольд, статья "Жесткие и мягкие математические модели"*

Примером крайней степени заблуждения является распространенный в математическом моделировании подход, когда система рассматривается как «черный ящик», то есть знания о ее структуре и внутренних принципах действия игнорируются. Вместо этого, рассматриваются входные и выходные параметры системы (разумеется, формализованные в численном виде), а затем между ними эмпирическим путем находится математическая закономерность. На основании этой закономерности в дальнейшем пытаются предсказать поведение системы при изменении параметров. Особенно удручающими выглядят попытки

> *"... можно ли математически определить биологическую систему, если мы не можем наделить эту модель системы самым важнейшим свойством живой системы: формированием потребности получить тот, а не другой результат, и определенный целью, которую обычно ставит перед собой биологическая система уже в самом начале формирования поведенческого акта?"*
> *ак. А.П. Анохин, статья "Принципиальные вопросы общей теории функциональных систем"*

такого предсказания, примененные к биологическим, социальным или экономическим системам.

Вопрос для размышления

Существуют ли математические модели, позволяющие достоверно предсказать поведение фондового рынка? Почему?

Разумеется, нельзя моделировать поведение системы, не рассматривая ее конструкции, которая состоит из элементов и правил взаимодействия между ними. Каждый элемент может быть, в свою очередь, также представлен как система. Такая декомпозиция должна продолжаться до тех пор, пока мы не придем к набору элементов, закономерности поведения которых нам известны с необходимой степенью достоверности. Семантические технологии предлагают отличный инструментарий для того, чтобы представить полученную модель в электронной форме, а затем работать с ней при помощи алгоритмов имитационного моделирования.

Стандартный способ использования модели, полученной в результате математического моделирования, состоит в следующем:

1. Реализация модели в программной среде, в процессе которой моделируемым объектам ставятся в соответствие программные объекты, а описывающие их показатели помещаются в переменные, связанные с этими объектами. Закономерности, включенные в модель, превращаются в вычислительные алгоритмы, связывающие между собой значения переменных.
2. Пошаговое выполнение модели, причем каждый шаг, как правило, соответствует выбранной единице времени. На каждом шаге происходит обновление значений переменных в соответствии с заложенными в модель вычислительными алгоритмами.
3. Интерпретация результатов моделирования: на основании итоговых значений переменных делаются выводы о том, как будет протекать моделируемый процесс в реальном мире.

Недостатки такого подхода очевидны, даже если отвлечься от уже рассмотренных проблем собственно математического моделирования:

1. Не все виды характеристик и отношений между объектами могут быть удовлетворительно или удобно описаны с помощью переменных и математических функций (особенно в экономике, где в моделировании участвуют, например, факторы человеческого поведения, или сложные логические условия).
2. Многие закономерности оказываются жестко запрограммированы в коде, реализующем алгоритмы моделирования.
3. Любая модель отражает определенную теорию, описывающую взаимодействие объектов. Если моделирование при помощи такой теории перестает давать удовлетворительные результаты, теорию приходится менять, что обычно приводит к необходимости полной перестройки всей модели.

Как изменится способ программного представления и использования модели благодаря семантическим технологиям? Для примера возьмем задачу, в которой поведение моделируемых объектов описывается сложным набором правил, имеющих логическую, а не математическую природу.

Пусть нас интересует прогноз загруженности муниципальных детских садов на несколько лет вперед. Модель для составления такого прогноза можно построить с использованием мультиагентного подхода. Этот подход состоит в реализации в компьютерной модели множества объектов, соответствующих реальным участникам процесса, или воспроизводящих их статистическое распределение. Исходными данными для моделирования являются:

- Набор детских садов, обладающих определенной емкостью для приема детей разного возраста,
- Набор жилых домов, входящих в участки, привязанные к каждому детскому саду,
- Семьи, проживающие в каждом из домов,

- Дети в каждой семье – реальные и прогнозные показатели.

Детские сады, жилые дома, и количество проживающих в них граждан – объективная информация, доступная из муниципальных баз данных. Конкретные семьи, количество детей в них, а также уровень дохода каждой семьи, могут генерироваться квазислучайным образом, в соответствии с известными из демографической статистики города кривыми распределения числа детей в семье, доходов семей, корреляций между доходом и районом проживания, доходом и количеством детей. Результатом процесса моделирования являются:

- Концептуальная модель, содержащая определения всех участвующих в моделировании типов объектов и их свойств;
- Набор экземпляров объектов с конкретными значениями свойств, соответствующий реальным или статистическим данным об объектах такого типа.

Полученная модель воплощается в компьютерное представление путем помещения составления онтологии, выражающей концептуальную модель, представления конкретных данных в соответствии с ней, и помещение всей этой информации в графовую базу данных (RDF Triple store) в виде определений классов и свойств объектов, а также экземпляров и значений свойств конкретных объектов.

Вопрос для размышления

Какая разница между тем, чтобы создать класс Java для описания поведения объекта в мультиагентном моделировании, и заложить в нем всю необходимую логику, и тем, чтобы описать этот объект и логику его поведения в онтологии?

Для запуска имитационного моделирования необходимо определить правила взаимодействия объектов на каждом шаге моделирования. Эти правила задаются в виде логических аксиом, использующих классы объектов и определения их свойств. Например, правило может утверждать, что семьи с определенным уровнем дохода предпочитают коммерческие детские сады муниципальным. Выражение «предпочитают» носит вероятностный характер, и может быть формализовано в

виде кривой распределения (число семей, выбравших муниципальный сад, в зависимости от уровня дохода). Кривая описывается математической функцией, или эмпирически – набором значений. В ходе моделирования каждая конкретная моделируемая семья «принимает решение» о выборе муниципального или коммерческого детского сада квазислучайным образом, в соответствии с этим распределением. Таким образом, в модель вносятся элементы нечеткой логики.

Другие правила могут определять предпочтения семей в плане того, в какой именно детский сад отдать ребенка. Например, если имеется сад, расположенный ближе к месту проживания семьи, чем тот, к которому она относится по прописке – с некоторой вероятностью можно допустить, что семья предпримет усилия для попадания именно в ближайший детский сад (снова в соответствии с функцией распределения, зависящей от дохода). При этом, если один ребенок данной семьи уже ходит в определенный детский сад, ее мотивация на то, чтобы в тот же сад попал и второй ребенок, резко повышается.

Все эти закономерности, как нетрудно видеть, имеют характер логических утверждений, применяющихся с определенной вероятностью, которая определяется математической функцией, зависящей от параметров объекта. Это позволяет внести в онтологическую модель в формализованном виде при помощи редактора онтологии как сами правила, так и задействованные в них расчетные формулы.

Такой подход позволяет легко расширять и изменять модель, добавляя в нее новые факторы и закономерности. Например, можно ввести обратную корреляцию количества коммерческих детских садов, и стоимости пребывания в них: рост спроса (увеличение числа «не пристроенных», не нашедших оптимального муниципального детского сада детей, или рост доходов населения определенного района) стимулирует сначала рост цены в существующих коммерческих детских садах, а затем – появление новых таких садов. Введение такой корреляции не требует изменения существующей части модели, и выполняется при помощи уже описанных выше средств моделирования. Таким образом, обеспечивается гибкость модели и возможность ее развития без какого-либо программирования.

Для пошагового симуляционного выполнения модели необходим программный компонент, который обеспечит последовательное применение всех заложенных в модели правил на каждом шаге моделирования, а также предоставит интерфейс для наблюдения за его ходом. Такой компонент создан в ходе наших работ, и успешно используется для решения практических задач.

Описанный подход может быть использован для решения задач во многих предметных областях. Приведенный пример касался муниципального управления; здесь можно выделить целый класс методически однородных задач, решаемых тем же способом (моделирование развития территорий, динамики спроса на любые муниципальные услуги).

В других сферах одним из наиболее перспективных направлений представляется моделирование поведения потребителей (или, шире – моделирование рынка), в интересах торговых и производственных компаний. Процесс принятия решения потребителем поддается моделированию при помощи описания баланса его потребностей и возможностей, а также ряда эмоциональных факторов (имидж бренда, плотность рекламных контактов и др.), формализуемых на основании эмпирических маркетинговых и психологических исследований и экспертных гипотез. Такое моделирование поможет, например, построить кривые зависимости объема продаж и прибыли от уровня цен, затрат на маркетинг и других параметров.

Аналогичным образом можно решать некоторые задачи моделирования сложных промышленных систем, состоящих из наборов взаимосвязанных узлов. Взаимосвязи узлов во многих случаях гораздо продуктивнее моделировать на логическом, а не физическом уровне. Например, обесточивание определенного трансформатора приведет к отключению всех питающихся от него устройств. Конечно, можно моделировать физику процесса – электрические цепи и происходящие в них электромагнитные явления; однако гораздо проще (и не менее продуктивно, с прикладной точки зрения) смоделировать эту зависимость при

помощи логической связи «подключен к», и логической аксиомы, утверждающей, что если объект А – подключен к – Б, то отключение Б приведет к отключению А.

Еще один интересный пласт задач – моделирование непрерывных сред. Такие задачи возникают, например, при моделировании геологических процессов. Классический способ моделирования таких сред состоит в ограничении модели определенной пространственной областью, задании краевых условий, разбивке моделируемой области на элементарные частицы объема (воксели), и математическом моделировании их физического взаимодействия (просачивание, давление и др.). Результативность такого подхода имеет известные границы. Семантический подход позволяет рассматривать моделируемую область одновременно на нескольких уровнях – макрообъектов (геологические формации, водоемы и др.), и микрообъектов (воксели – единицы пространства). Часть закономерностей взаимодействия объектов хорошо описывается на микроуровне, другая часть – на макроуровне. Их объединение в одну модель, как правило – интегрирующую данные нескольких принципиально разных видов исследований, должно дать возможность получать куда более качественные результаты. Краевые условия при этом подразумевают, что моделируемая область пространства окружена средой, также имеющей определенные свойства и взаимодействующей с моделью, но только на макроуровне. Таким образом, моделируемый участок реальности не отделен от окружения умозрительной «стеной».

Важным частным случаем описанных здесь задач являются модели пространственно распределенных систем. Ничем не отличаясь с точки зрения методологии моделирования, они имеют специальные требования по представлению получаемых результатов, что должно отразиться в интерфейсе управления моделью.

7.3. Моделирование наборов показателей на примере показателей защищенности

На примере показателя EBITDA мы уже демонстрировали тот факт, что при моделировании наборов показателей, описывающих функционирование какой-либо системы, легко получить не релевантные результаты. Приведем пример корректного построения набора показателей, и заодно продемонстрируем возможные «ловушки», возникающие на этом пути.

В управлении безопасностью, как и во многих других областях, принято описывать состояние управляемой системы при помощи ключевых показателей. На языке показателей безопасность обычно определяется как величина, обратная риску наступления неблагоприятных событий. Риск же является произведением возможного ущерба на вероятность его возникновения. Управление рисками составляет достаточно зрелую дисциплину, применяемую во многих предметных областях. Ключевые показатели риска (KRI) обычно задаются для каждого вида факторов влияния, существенных для системы. Возьмем в качестве примера предельно простой случай технологической системы, и опишем для нее набор KRI.

Рис. 57. Моделируемая система

Пусть наша система состоит из свечи, над пламенем которой расположена ось с лопастями из фольги. Будем считать, что система выполняет свою функцию, пока восходящий поток газа от пламени свечи вращает ось. Будем принимать в расчет два фактора влияния для этой системы:

- Если температура воздуха превысит 700С, парафин расплавится, свеча потеряет форму и потухнет.
- Если скорость ветра превысит 0,5 м/с, свечу задует.

Введем также следующие исходные положения:

- Мы не имеем возможности предсказывать изменение температуры воздуха и скорости ветра, то есть любое неблагоприятное событие наступает для нас полностью неожиданно.
- Ущерб в результате наступления любого из событий примем равным 1 рублю.
- Из статистики наблюдений известно, что в 75% случаев система выходит из строя из-за расплавления свечи, а в 25% случаев – из-за задувания.
- Известно, что вероятность расплавления свечи составляет 1 $час^{-1}$ (событие происходит 1 раз в час), вероятность задувания – 0,33 $час^{-1}$ (1 раз в 3 часа).
- Исчерпанием ресурса свечи пренебрежем.

Имея эти исходные данные, оценим показатели риска. Риск равен произведению ущерба на вероятность его наступления, то есть риск расплавления будет равен $R_р$=1, а риск задувания – $R_з$=0,33. Общий риск для системы предлагаем рассчитать, как сумму двух указанных рисков, что составит $R = R_р + R_з = 1,33$ руб/час. Такого подхода придерживаются, например, «Методические рекомендации по оценке рисков на железнодорожной инфраструктуре ОАО «РЖД». В некоторых других источниках предлагается при группировке частоты аварий с разными причинами использовать весовые коэффициенты, соответствующие относительному «вкладу» причины в общую статистику отказов. По такой

методике, суммарный риск составил бы $R=0,75\ R_р+ 0,25\ R_з=0,83$, что, на наш взгляд, неверно.

Примем этот уровень риска за «естественный», и опишем при помощи количественных показателей безопасность системы. Для этого нужно перейти от показателей риска к показателям защищенности системы, и здесь возникает риск (уже для нас, как авторов методики расчета) существенной потери информации и искажения картины. В общем случае, степень защищенности должна быть обратна вероятности реализации риска, то есть описывать вероятность его не-реализации. Для этого авторы книги "Управление ресурсом эксплуатации высокорисковых объектов" (под ред. Н.А. Махутова) предлагают использовать формулу $S = 1 – R$, где S – защищенность; однако, в этом случае шкала S сильно зависит от порядка значения R, которое, к тому же, должно быть нормировано к 1. Использовать выражение $S = 1/R$ на практике тоже чаще всего неудобно из-за того, что значение показателя оказывается не нормированным. В результате этого, например, специалисты ФГУ ВНИИ ГОЧС (Глебов В.Ю., Головина Н.С. Методический подход к оценке защищенности критически важных для национальной безопасности объектов Российской Федерации от угроз техногенного, природного характера и террористических проявлений) рекомендуют использовать в качестве метрики защищенности степень реализации мероприятий по защите, предусмотренных требованиями. Здесь происходит важный смысловой сдвиг: защищенность становится характеристикой мероприятий по защите и теряет привязку к риску, который обеспечивал физический смысл ее значения.

Посмотрим, что происходит при реализации этого метода. Пусть для нас защищенность от фактора риска равна 0, если этот риск равен «естественному», то есть в отсутствие проводимых мероприятий; и равен 1, если риск сведен к 0, то есть объект абсолютно защищен от данного риска. Общую защищенность будем определять как среднее арифметическое от защищенности по всем направлениям, с весовыми коэффициентами, соответствующими исходной статистике причин аварий. В начальном состоянии системы защищенность от обоих факторов равна 0,

как и общая защищенность системы. Предположим, мы установили защитный экран, который вдвое снижает частоту события «задувание свечи», то есть соответствующий риск становится равным 0,16 руб/час. Общий риск будет равен 1,16 руб/час, защищенность от задувания – 0,5, а общая защищенность S = 0,75*0 + 0,25*0,5 = 0,125. Обратим внимание на то, что защищенность при этом становится безразмерной величиной. Далее, предположим, что вместо экрана мы установили систему кондиционирования воздуха, которая сделала невозможным подъем температуры до 70 градусов. Тогда риск расплавления становится равен 0, общий риск – 0,33 руб/час, защищенность от расплавления – 1, а общая защищенность S = 0,75*1 + 0,25*0 = 0,75. Полученные результаты, казалось бы, соответствуют здравому смыслу. Проблемы начнутся в тот момент, когда риски начнут увеличиваться по сравнению с «естественными» значениями. Предположим, в атмосфере увеличилась турбулентность, и вероятность задувания свечи повысилась до 0,66 час$^{-1}$ (1 раз в полтора часа). Защищенность, в соответствии с нашим алгоритмом, останется равной 0, или примет отрицательное значение (в зависимости от методики вычисления). Общий риск, однако, возрастет до 1,66 руб/час (в отсутствие мероприятий). Более того, если кондиционер в помещении был установлен, защищенность будет равна 0,75, а риск составит 0,66. Показатель риска будет в два раза выше, чем при исходном значении турбулентности, а на показателе защищенности это никак не отразится: он будет по-прежнему высок. Увидев такой показатель на панели, руководитель будет считать, что система находится в относительной безопасности, и не нуждается в дальнейшей защите.

Ситуация	**Риск**	**Защищенность**
Без мероприятий (низкая турбулентность)	1,33	0
Установлен экран	1,16	0,125
Установлен кондиционер	0,33	0,75
Без мероприятий (высокая турбулентность)	1,66	0
Установлен кондиционер	0,66	0,75

В таблице, куда мы свели результаты нашего эксперимента, цветами выделены разные ситуации, характеризующиеся различным уровнем риска, но одинаковой «защищенностью».

Более того – при этом происходит и потеря информации. В классической теории информации (Голдман и др.) количество (не объем!) информации является мерой устранения неопределенности. Применим этот принцип к нашему случаю. Показатель риска полностью информативен – он устраняет неопределенность относительно того, какими будут средние материальные потери в единицу времени. Показатель защищенности, казалось бы, должен устранять неопределенность относительно того, насколько система устойчива к воздействию неблагоприятных факторов. Однако мы видим, что для двух разных случаев – с низкой и высокой турбулентностью атмосферы – значение защищенности одно и то же. Следовательно, имея в своем распоряжении значение этого показателя, мы не устраняем неопределенность относительно способности системы сопротивляться внешним факторам – просто потому, что показатель описывает только защитный потенциал самой системы, безотносительно внешних условий, которые могут сложиться таким образом, что наши средства защиты окажутся абсолютно бесполезными. Значит, заключающееся во введенном нами показателе «защищенность» количество информации меньше, чем в показателе «риск».

Каково же решение проблемы? Оно состоит в том, чтобы понятие защищенности не теряло прямой связи с риском. Приняв S=1/R, получим следующие результаты:

Ситуация	**Риск**	**Защищенность**
Без мероприятий (низкая турбулентность)	1,33	0,75
Установлен экран	1,16	0,86
Установлен кондиционер	0,33	3
Без мероприятий (высокая турбулентность)	1,66	0,6
Установлен кондиционер	0,66	1,5

При таком варианте защищенность имеет размерность – час/руб, и физический смысл – она соответствует среднему времени, в течение которого мы теряем 1 рубль стоимости активов вследствие неблагоприятных событий. Очевидно, что чем лучше объект защищен, тем больше времени будет требоваться на накопление потерь. Эти результаты хорошо согласованы, но, к сожалению, значение защищенности оказывается не нормированным, и при риске, стремящемся к нулю, стремится к бесконечности – что делает его крайне неудобным для использования в панелях индикаторов.

Однако, этим проблемы оценки рисков не исчерпываются. Вернемся к самому началу нашего рассуждения. Мы определили риск через вероятность наступления события, несущего негативные последствия. Что же делать в случае, когда эта вероятность принципиально не может быть определена? Хрестоматийным примером является ситуация с защищенностью золотого запаса США, хранящегося в Форт Нокс. За всю историю не произошло ни одного ограбления этого хранилища, и при всей изощренности воображения аналитиков невозможно представить себе сколько-нибудь достоверную ситуацию, при которой оно могло бы удаться. Следовательно, риск ограбления этого объекта стремится к нулю, а его защищенность – к бесконечности. Однако, это совсем не означает, что можно ослабить охрану данного объекта. Таким образом, в этом случае становится очевидным, что значения показателей не могут быть единственным основанием для принятия управленческих решений.

Наш вывод состоит в том, что ключевые показатели риска и безопасности могут быть полезными, но при этом необходимо:

а) четко соблюдать и обосновывать наличие у них физического смысла,

б) исключать произвольность методики вычисления показателей,

в) исключать их нормирование с потерей смысла и размерности,

г) при принятии управленческих решений принимать во внимание, что следствием этих решений может быть изменение «правил игры», которое может

повлечь перестройку в том числе системы показателей как таковой, правил их определения, а не только значений.

7.4. Время в семантических моделях

Важным вопросом, который мы не можем обойти в нашем рассказе, является отражение времени в семантических моделях. Использование семантической модели всегда подразумевает, как минимум, ее эволюцию; при этом естественным требованием является возможность просмотра состояния системы на любой момент времени, или, по крайней мере, истории изменения определенных объектов.

Если модель используется в целях имитации каких-либо процессов, происходящих в реальном мире – нужен механизм отсчета времени, «часы», в соответствии с ходом которых будет изменяться состояние модели.

Наконец, если в модели просто отражаются какие-либо события, сопровождающиеся изменением состояния ее элементов – нужен механизм указания времени каждого события.

Однако никаких специальных технических средств для отражения временно́го компонента семантические технологии не предлагают, за исключением того, что свойства-литералы могут иметь тип значений «дата» или «дата и время». Остается компенсировать этот недостаток за счет методики моделирования, которая будет отражать ход времени и эволюцию компонентов модели.

John Sowa в книге «Knowledge representation. Logical, philosophical and computational foundations» выделяет два вида объектов по способу существования во времени: продолжающиеся (continuant) и происходящие (occurrent). Первые характеризуются сохранением некой целостности, не зависящей от изменения состава и свойств объекта. Например, человек на протяжении своей жизни обменивается веществом и энергией с окружающей средой, изменяет свои физические и социальные характеристики, но остается тем же самым существом. К «происходящим» объектам можно отнести события, состояния. Как правило,

у них имеется некая логика развития, которую можно описать при помощи фаз, называемых также «временны́ми частями». Таким образом, если сам по себе человек является «продолжающимся» объектом, его жизнь можно представить в виде совокупности «происходящих» объектов, выделение которых зависит от цели моделирования.

Техническую реализацию работы с временны́ми частями в OWL предлагает, в частности, стандарт ISO 15926. Суть подхода состоит в разделении объектов на «временно́е целое» (Temporal Whole) и «временные части» (Temporal Parts). Каждая часть – это фаза существования объекта, характеризующаяся начальным и конечным моментом ее актуальности. Временно́е целое характеризует ту сущность, которая соответствует объекту на всем протяжении его жизненного цикла. Если мы представим, вернувшись к рис. 11, что собака откусила человеку палец, то в модели это будет отражено так: временно́е целое (человек, объект A1) будет иметь две временных части – A11, в состав которой входит палец, и A12, в составе которой пальца уже нет. Соответственно, свойства объекта как целого приписываются временно́му целому, а те свойства, которые могут изменяться, включая компонентный состав объекта – временным частям. На практике, очевидно, изменяться могут практически любые свойства объекта, что ведет к перегрузке временных частей свойствами; необходимость создавать новую временную часть при каждом существенном событии ведет к неконтролируемому росту объема модели. Это снижает ее практическую ценность, поскольку человеку крайне сложно разобраться в такой модели, а программным алгоритмам – выполнять на них вычисления. Таким образом, методологически правильная идея оказывается трудно применимой на практике.

Как же отражать время в моделях, чтобы это не приводило к их чрезмерному усложнению? На наш взгляд, решением является создание класса «событие», члены которого имеют обязательное свойство – момент или период времени, когда событие произошло. Здесь и пригодится возможность создавать свойства с типом значения «дата и время». Подклассы этого класса должны определять конкретные

виды событий (конечно, иерархии типов событий можно организовать как угодно. После того, как событие произошло, в модели должен появляться соответствующий объект – таким образом будет формироваться «журнал изменений» модели. Событие может иметь ссылки на те объекты, состояние которых изменилось в результате него, и описание этих изменений. Сами же объекты будут немедленно менять свое состояние, и, таким образом, всегда соответствовать «актуальному» представлению модели.

При такой организации отражения времени мы реализуем аналог его восприятия в реальности: в каждый момент времени система имеет одно, определенное состояние (мы не можем вернуть «прошлогодний снег»). Однако мы помним о том, какой была система раньше – набор наших объектов класса «событие» образует такие «воспоминания», которые можно проматывать в любую сторону.

Важно, что для семантических вычислений и выводов используется, как правило, именно текущее состояние системы – а его мы ничуть не усложнили введением «журнала событий». В то же время, этот «журнал» дает нам возможность обратиться к истории, и узнать, как эволюционировало состояние того или иного объекта.

Оба подхода – и работа с временны́ми частями, и запись «журнала событий» – имеют право на существование. Каждый из них имеет свои преимущества и недостатки, опираясь на которые, можно выбрать наиболее подходящий способ в каждом конкретном случае.

7.5. Способы группировки в онтологическом моделировании

Как мы видели, в арсенале средств семантических технологий основным способом группировки сущностей является создание классов и включение в них индивидуальных объектов. Существуют и другие способы, основным из которых является выражение принадлежности индивидуального объекта к какой-либо

группе путем представления группы в виде другого индивидуального объекта, и создания связи между ними при помощи значения атрибута одного из объектов. Для того, чтобы разобраться, какой способ в каком случае следует использовать, нужно сначала рассмотреть принципы группировки объектов в концептуальном моделировании.

John Sowa в процитированной выше работе выделяет четыре способа группировки сущностей в концептуальной модели, соответствующие разным случаям практического применения и различным логическим теориям. Каждый из этих способов может быть определенным образом отражен средствами OWL. Перечислим эти способы и соответствующие им способы формализации в онтологиях:

Способ группировки в концептуальной модели	**Суть способа**	**Варианты отражения в OWL**
Коллекции	Коллекция – составной объект, включающий разнородные, не обязательно исчисляемые, не обязательно четко разделяемые части. Пример: куриный суп с клецками. Логическая операция: объект является частью коллекции.	Создание коллекции как самостоятельного объекта, указание его связи с каждой частью при помощи атрибутов.
Множества	Множество – совокупность однородных, исчисляемых объектов. Пример: собаки. Операции: объект является членом множества, множество имеет подмножество.	Классы и подклассы, принадлежность объектов к классам.

Типы	Тип – произвольная группа объектов, которую аналитик счел нужным выделить по какому-либо основанию. Пример: опасные объекты. Операции: объект имеет тип, тип имеет подтип.	Возможно использование как классов, так и связь при помощи атрибута с объектом, описывающим тип.
Категории	Типы, используемые с целью классификации. Пример: объект I категории опасности. Операции: объект относится к категории, категория имеет подкатегорию.	Возможно использование как принадлежности к классам, так и атрибутов-ссылок на категорию.

Заметим, что разница между типами и категориями оказывается достаточно условной, и отражает скорее наличие разных математических теорий в основании, нежели смысловое различие.

Типы могут быть абстрактными, то есть принципиально не иметь относящихся к ним индивидуальных объектов (пример: единорог). Хотя в любом случае полезно помнить, что мы объединяем в группы не сами объекты реального мира, а их образы в модели – таким образом, в каком-то смысле все содержание выделяемых в модели групп является абстрактным. Пустые множества также возможны, но их прагматика в онтологических моделях менее очевидна.

Поскольку тип подразумевает некую идею, лежащую в основе его выделения, каждый тип характеризуется *интенсионалом* и *экстенсионалом*, о которых мы уже упоминали в начале пособия. Интенсионал – это «идея», обеспечивающая смысл выделения типа, которую заложил в его определение аналитик. Экстенсионал – набор объектов, соответствующих данному типу. Экстенсионал разных типов может совпадать: например, типы «персоны, имеющие действующий трудовой договор с ООО «Альфа»» и «люди, работающие в офисе компании «Альфа»» могут

совпасть по набору соответствующих объектов, хотя условие выделения группы (интенсионал) здесь различно.

С практической точки зрения, можно выделить две основных цели группировки объектов в онтологии:

I. Сообщение фактов об общих качествах, характеристиках объектов (но не обязательно – о значениях характеристик);
II. Описание характеристик группы объектов, как целого (характеристики группы зависят от ее состава).

Сравним эти цели в таблице.

	Сообщение фактов об общих качествах объектов	**Описание характеристик группы объектов, как целого**
Способ получения выводов	Получать выводы о свойствах каждого элемента исходя из свойств группы	Делать выводы о целом исходя из знания о его частях
Однородность группы	Объекты являются однородными в каком-то отношении	Объекты могут быть разнородными
Возможность указать все объекты группы	Обычно присутствует, так как нам интересны именно конкретные объекты	Может отсутствовать: достаточно знать *min* элементов, которые включены в группу, но не обязательно знать каждый элемент (пример: лес состоит из деревьев)

Смысл группы	Группа – общность объектов	Группа – самостоятельный объект, взаимодействующий с другими частями модели

Исходя из этого, в онтологической модели целесообразно применять следующие способы выражения групп объектов.

Для выражения группировки однородных объектов – принадлежность к классу. Например, к классу «товар» относятся все товары, рассматриваемые в нашей модели. Мы знаем, что с товарами можно выполнять определенные операции, они имеют некоторые характеристики, и благодаря этому можем планировать практическую деятельность – закупку, хранение, перевозку, продажу товаров.

Для выражения единого объекта, состоящего из других объектов – индивидуальный объект. Например, лес состоит из деревьев, но мы не знаем каждого дерева в отдельности; тем не менее, мы знаем общие свойства деревьев (как этого конкретного леса, так и вообще), и это дает нам возможность планировать, например, вырубку и переработку леса. «Дерево» в этом случае может стать классом объектов, а конкретные породы деревьев – «сосна», «береза» – индивидуальными объектами, обладающими определенными свойствами. Индивидуальный объект «лес» (точнее, например, «квартал №123») будет связан с этими объектами некоторыми отношениями. Отношения сами по себе могут быть индивидуальными объектами класса «Состав», и нести информацию не просто о том, что лес состоит из сосен, но и о том, каков процентный состав сосен в нем.

Заметим, что оба подхода могут быть скомбинированы.

Нет принципиального отличия между ситуацией, когда целое состоит из однородных частей, и не однородных. Например, на каком-то уровне упрощения мы можем утверждать, что «общество состоит из людей». Детализируя модель, мы обнаружим, что общество состоит еще и из экземпляров социальных институтов – семей, местных общин, трудовых коллективов и др. На первом уровне детализации

мы будем пытаться объяснить какие-то процессы или характеристики общества, опираясь на знания о том, каковы люди, его составляющие; на втором уровне – подключим к этому объяснению и характеристики институтов. Автомобиль – целое, состоящее из разнородных деталей, и поведение автомобиля может быть смоделировано исходя из знаний об этих деталях, и правилах их взаимодействия. Лес – целое, состоящее из однородных элементов, и мы точно также можем утверждать что-то о лесе, имея некоторую информацию о деревьях. И то, и другое – *системы*, и нет принципиальной разницы, однородны ли их элементы.

Перейдем к вопросу о том, как именно следует выражать отношение объекта к группе. Понятно, что если группа формализована в модели как класс, то объект должен стать членом этого класса (используется предикат rdf:type), а если группа является самостоятельным объектом – то объект связывается с ним значением некоторого атрибута-связи, который необходимо определить (например, «является частью»). Сразу заметим, что использование машины логического вывода позволяет автоматически создавать классы на основании наличия связей (например, автоматически выделить класс «детали автомобиля» на основании того, что некоторая деталь связана с некоторым автомобилем связью «является частью»).

Выше мы говорили о том, что класс следует выделять только в тех случаях, когда мы намерены делать выводы о свойствах входящих в него объектов на основании принадлежности к классу. Поясним это на примере.

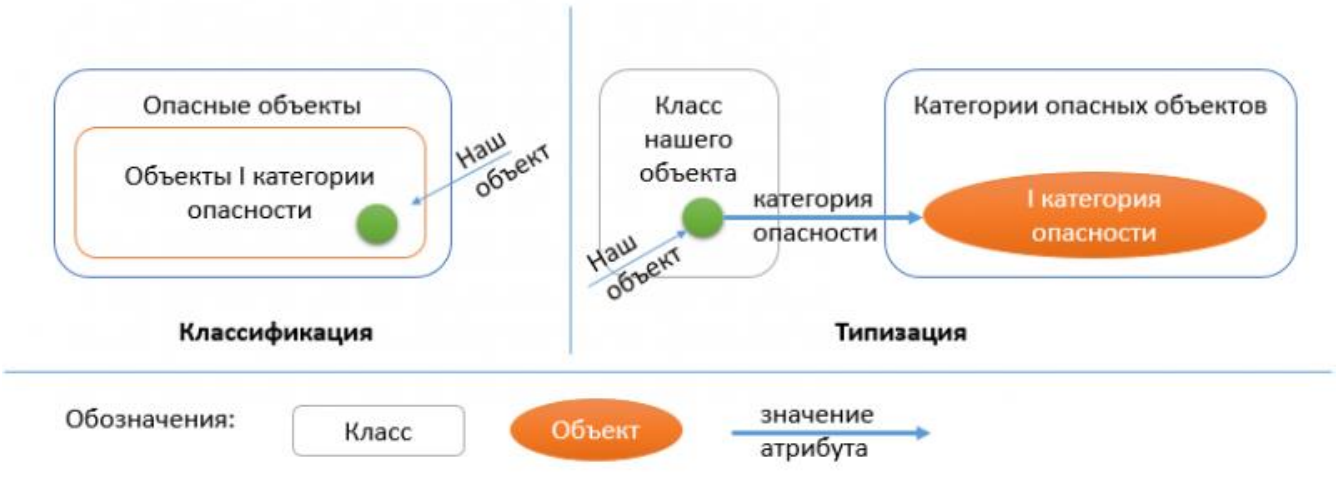

Рис. 58. Классификация и атрибутирование

Предположим, у нас имеется задача выразить отношение объекта «наш объект» к группе «I категория опасности». В этом случае трудно применить наш предыдущий критерий, так как способ использования не следует однозначно из контекста задачи: с одной стороны, скорее всего, мы будем строить некие регламенты обработки событий, опираясь на тот факт, что объект относится к I категории опасности, то есть будем делать выводы о конкретном объекте исходя из знаний о классе; с другой стороны, вполне может оказаться, что совокупность объектов I категории опасности выступает самостоятельным объектом – например, в качестве способа группировки в отчете, или объекта, о котором делаются утверждения в нормативной документации. При отсутствии дополнительной информации, первый способ (классификация) кажется предпочтительным, однако контекст задачи может давать предпосылки к использованию второго способа.

В таких случаях мы предлагаем использовать простой технический критерий для выбора подходящего варианта. На левой части картинки показана ситуация, когда «I категория опасности» является подклассом класса «Опасные объекты». Этот способ выражения связи мы назовем *классификацией*. На правой части картинки изображена ситуация, когда «I категория опасности» является индивидуальным объектом класса «Категории опасных объектов», а связи выражается значением свойства «категория опасности» объекта «наш объект».

Целесообразность использования первого варианта с технической точки зрения доказуема в случае, если:

- от того, что объект принадлежит к данному классу, зависит состав его атрибутов;
- есть причина сделать рассматриваемую категорию именно классом, не связанная с данным объектом (атрибуты должны появиться у каких-то других объектов, которые будут ею классифицированы);

- рассматриваемые категории образуют иерархию, и возможна дальнейшая детализация данной категории (например, I категорию опасности можно разделить на подкатегории).

В остальных случаях следует выбирать второй вариант. Он имеет неоспоримые преимущества в случае, если у категории имеются собственные атрибуты (класс в OWL лишен собственных значений атрибутов, и это очень неудобно). Кроме того, вполне вероятно, что в варианте с атрибутированием будет удобнее строить выборки, фильтры и т.п.

7.6. Применения онтологических моделей

Мы рассмотрели основные концепции онтологического моделирования, и рассмотрели некоторые из его применений. Приведем теперь полный перечень способов использования онтологических моделей и классы задач, для решения которых эти способы пригодны.

Мы уже упоминали *имитационное моделирование*. Его суть состоит в следующем: модель воспроизводит в существенных отношениях какой-либо фрагмент реальности. Мы имеем возможность запустить процесс выполнения модели, в ходе которого ее элементы взаимодействуют между собой таким же образом, как их прототипы в реальном мире.

Главным требованием к имитационной модели является возможность *трансляции* результатов. При построении модели мы неявно определили правила, по которым объекты реального мира отображаются на объекты модели. Если мы выполним в реальном мире и в модели один и тот же процесс, рассматриваемая система окажется в определенном конечном состоянии. Трансляция результатов будет обеспечена в том случае, если отображение конечного состояния системы в реальном мире на модель по установленным нами изначально правилам, даст в точности то же состояние модели, в которое она пришла сама в результате симуляции.

Если выразиться проще, условие трансляции обеспечивает практическую пригодность модели для прогнозирования того, как будет проходить процесс в реальном мире. Понятно, что любая модель обеспечивает трансляцию результатов только при соблюдении определенных условий (например, модель, которую мы строили по рис. 11, не учитывает возможности вмешательства в сцену кого-либо еще). Эти условия должны быть такими, чтобы вероятность их нарушения при протекании процесса в реальном мире являлась допустимой с точки зрения практических задач моделирования.

Имитационное моделирование пригодно для решения широкого круга задач, связанных с моделированием любых сложных систем: социальных, экономических, биологических, технических, физических. Принимая решение о реализации имитационной модели при помощи семантических технологий, необходимо проверить, значимы ли их преимущества в данном случае.

Вопрос для размышления

Можно ли решить при помощи имитационного моделирования физическую задачу многих тел – например, предсказать траекторию движения астероида в Солнечной системе? Насколько рационально использовать именно эту технологию?

Вторым применением семантических моделей является *поддержка принятия решений*, создание экспертных систем. В самом деле, мы можем записать в виде логических утверждений правила поведения объектов, и на этом основании предсказывать возможные последствия тех или иных решений и действий. Предположим, мы создаем модель, предназначенную для выбора медицинских исследований, которые следует назначить пациенту. В ней могут содержаться такие утверждения: «Если пациент жалуется на сильную головную боль, ему необходимо назначить УЗИ сосудов шеи и головного мозга». Если такая модель будет реализована в компьютерной программе, с которой работает сотрудник регистратуры, он сможет выбрать симптомы, с которыми обратился пациент, и получить распечатку назначений на обследования. С этими результатами

пациент пойдет на прием к врачу. Конечно, подобная система будет содержать множество вариантов ветвления, которые потребуют проведения небольшого интервью с пациентом в зависимости от того, какие симптомы он называет. Важно отметить, что создает модель человек с более высокой квалификацией (врач), чем тот, кто ее использует (регистратор) – таким образом, модель служит средством переноса знаний.

Разновидностью подобных систем можно считать программы, оперативно реагирующие на те или иные события без участия человека. Например, система контроля за каким-либо технологическим оборудованием может содержать следующие утверждения: «Если давление в трубопроводе Б1 превышает Х кПа, необходимо открыть клапан Б2». Понятно, что подобные алгоритмы можно запрограммировать и без всякого семантического моделирования. Преимущества его использования в данном случае носят исключительно технологический характер: они позволяют облегчить процесс переноса знаний инженера, способного сформулировать такие правила, в инструкции, которые выполняет контрольное оборудование.

Вопрос для размышления

В приведенном примере система только применяет знания, заложенные в нее человеком в готовом виде. Можно ли реализовать в такой системе элементы получения новых знаний?

Другой разновидностью такого применения семантических моделей можно считать автоматическую генерацию инструкций и регламентов для людей. Получив при помощи имитационного моделирования (первый способ) или на основании знаний экспертов (второй способ) определенные выводы о том, как нужно действовать в тех или иных ситуациях, мы можем автоматически экспортировать из модели набор утверждений, которые не требуют от человека принятия никаких решений, а просто говорят ему о том, что если произошло событие А – то нужно выполнить действие Б. Таким образом легко генерировать простые и понятные инструкции для

персонала. В частности, так можно преобразовывать формальные описания бизнес-процессов в инструкции для каждого участника процесса.

Третье применение семантических моделей состоит в решении задачи *переноса информации* между различными средами. В этом случае семантическая модель играет роль «словаря», или общего языка, понятного всем участвующим в обмене. Внутри себя каждая система (источник, приемник) может хранить информацию в любом естественном для нее представлении. Экспортируя эту информацию наружу – система должна представить ее в обобщенном виде, для чего используется «словарь», представляющий собой элементы общепринятой информационной модели.

Важным применением семантических моделей, использующим элементы всех трех перечисленных выше подходов, является решение оптимизационных задач. Как правило, они сводятся к составлению оптимальной с экономической точки зрения программы действий в какой-либо сфере. Например, на территории предприятия имеется 100 трансформаторов, и в текущем году выделены деньги на профилактический ремонт 10 из них. Каким образом следует потратить эти деньги, чтобы минимизировать вероятность аварий в течение следующего года?

Качество решения таких задач напрямую зависит от детальности и адекватности составления модели. Простейшую модель для решения указанной задачи можно составить в Excel без всякой семантики: построить список трансформаторов с указанием года последнего ремонта каждого из них, отсортировать по этому столбцу, и взять 10 самых «заброшенных» трансформаторов. Очевидно, однако, что это решение не является оптимальным, поскольку не учитывает степень важности трансформаторов для технологических процессов предприятия, возможный ущерб от аварии на каждом из них, нагрузку и степень износа каждого трансформатора, данные контроля, выполненного электриками. Следующим соблазном чисто математического упрощения задачи является сведение всех перечисленных параметров в один числовой коэффициент, и ранжирование трансформаторов по нему. Например, мы можем оценить степень

важности каждого трансформатора по шкале от 0 до 100, и придать этому параметру весовой коэффициент 0.2. Аналогичным образом «оцифровав» все остальные параметры, и раздав им весовые коэффициенты, общая сумма которых будет равна 1, получим синтетический «коэффициент потребности в ремонте» для каждого трансформатора, нормированный на единицу. Отсортировав трансформаторы по нему, мы получим более качественное решение, чем при сортировке по году последнего ремонта, однако оно все еще будет далеко не оптимальным. Например, возможны сочетания факторов, которые делают потребность в ремонте критически важной, при чем значение остальных параметров перестает играть роль. Так, если наш трансформатор обслуживает хранилище токсичных отходов, при обесточивании которого может произойти их выброс, что приведет к экологической катастрофе и полной остановке работы предприятия – очевидно, что этот трансформатор должен ремонтироваться «вне конкурса», при достижении им 50% выработки ресурса. Таких правил может быть множество; их нельзя описать в математической модели, но можно описать в семантической.

Для решения оптимизационных задач используется сочетание семантических и математических моделей. Семантическая модель используется для того, чтобы отразить все богатство атрибутов и связей элементов моделируемой системы. На этой модели проводятся логические и математические вычисления, которые позволяют найти оптимальное решение. Как правило, такое моделирование является скользящим: модель постоянно обновляется за счет поступления новых сведений о состоянии моделируемой системы, и, соответственно, автоматически пересчитывается оптимальное решение, что приводит к коррекции программы действий.

Наконец, еще одним применением онтологических моделей является помощь человеку в *организации знаний*. Хорошо известна проблема, состоящая в том, что по мере роста сложности информационной инфраструктуры любого предприятия, она теряет управляемость. Резко повышается время, необходимое на поиск той или

иной информации, что приводит к утрате гибкости в принятии решений. Иногда объем накопившейся информации таков, что проводить в нем поиск вручную принципиально невозможно. В таких случаях единственным вариантом решения проблемы является создание семантической модели той среды, в которой функционирует предприятие, с привязкой к каждому элементу этой модели ссылок на связанные с ним информационные ресурсы. Такая модель играет роль «оглавления», по которому сотруднику легко найти любую нужную ему информацию.

Представим себе, что компания обслуживает сложный технологический объект – например, трубопровод. В случае аварии на каком-либо узле этого трубопровода инженерам необходимо быстро найти сведения о типе установленного там оборудования, его возможных аналогах, наличии этих аналогов на складе или у поставщиков, информацию о датах и результатах последних плановых ремонтов этого узла, сведения о предыдущих инцидентах на нем. Даже при самой лучшей организации информационных систем на предприятии невозможно ожидать, что все эти сведения будут доступны в рамках одной программы (ТОиР[2]): сведения об аналогах и их наличии на складе/у поставщиков явно выходят за рамки ее сферы ответственности. Значит, в семантической системе к модели данного узла должен быть привязан:

- состав (типы) использованного там оборудования,
- сведения о конкретных единицах оборудования, которые установлены на этом узле сейчас, и были установлены ранее,
- сведения о взаимосвязях данного узла (что подключено к нему, к чему подключен он),
- ссылки на документацию (проектную, исполнительскую, ремонтную) по узлу и всем его компонентам,

[2] Класс информационных систем управления Техническим Обслуживанием и Ремонтом

- ссылки на информацию о закупках оборудования такого типа,
- ссылки на информацию системы ТОиР с историей обслуживания этого узла и инцидентов на нем, и т.д.

Обладая таким «оглавлением», инженер сможет быстро найти все необходимые сведения, и запланировать ремонтные мероприятия.

***

Мы кратко рассмотрели назначение, идеологию и основные технические средства семантического моделирования. Возможные практические применения этой технологии настолько разнообразны, что не представляется возможным сделать сколько-нибудь полный их обзор в рамках одной книги. Мы сознательно не углублялись в детали синтаксисов, развернутые примеры применения и прочие технологические подробности – эту информацию легко можно найти в спецификациях и руководствах, доступных в Интернете. Нашей главной задачей было сформировать у читателя представление о возможностях семантического моделирования; надеемся, эта информация принесет осязаемую пользу в вашей практической деятельности.