МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

Е.В. СИМОНОВА

МОДЕЛИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Часть II. Использование онтологии для моделирования сложных адаптивных систем

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С.П. Королева» в качестве учебного пособия для обучающихся по основной образовательной программе высшего образования по направлению подготовки 09.04.01 Информатика и вычислительная техника

САМАРА
Издательство Самарского университета
2022

УДК 004.9(075)
ББК 32.97я7
С375

Рецензенты: канд. техн. наук, доц. Л.С. Зеленко,
д-р техн. наук, проф. С.В. Смирнов

Симонова, Елена Витальевна

С375 Моделирование информационных систем. Часть II. Использование онтологии для моделирования сложных адаптивных систем: учебное пособие / *Е.В. Симонова*. – Самара : Издательство Самарского университета, 2022. – 112 с. : с ил.

ISBN 978-5-7883-1758-8 (ч. 2)
ISBN 978-5-7883-1759-5

Учебное пособие предназначено для использования при изучении курса «Моделирование информационных систем». Пособие посвящено рассмотрению онтологического подхода к представлению и хранению знаний, позволяющего описать любую разнородную, сколь угодно сложную предметную область. Рассматриваются онтологии различного уровня, применение онтологий в Semantic Web, языки представления онтологий, модели онтологии, программные инструменты для работы с онтологиями.

Предназначено для обучающихся по направлению подготовки 09.04.01 Информатика и вычислительная техника.

Подготовлено на кафедре информационных систем и технологий.

УДК 004.9(075)
ББК 32.97я7

ISBN 978-5-7883-1758-8 (ч. 2)
ISBN 978-5-7883-1759-5

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Современные автоматизированные системы управления стремительно усложняются, что вызвано растущими требованиями бизнеса, высокой неопределенностью и динамикой изменения спроса/предложения в новой глобальной экономике знаний. Одним из подходов к решению данной проблемы является создание интеллектуальных систем поддержки принятия решений в реальном времени. В этой связи в настоящее время резко возросла потребность в представлении, визуализации, формализации, интеграции, хранении и повторном использовании больших объемов знаний и данных в различных областях деятельности.

Основная проблема заключается в том, что не все знания можно легко описать и запрограммировать, так как они обычно являются разнородными, многосвязными, неполными, могут содержать некорректную информацию, быть как декларативными, так и процедурными, связанными не только иерархическими, но и сетевыми структурами, и т.д. Поэтому важнейшие и специфические знания для принятия решений во многом остаются неформализованными, что затрудняет использование автоматизированных систем для управления сложными объектами и процессами.

Другой трудностью является жесткое кодирование данных в тексте программы, изменение которых ведет к перепрограммированию всей системы. Поэтому необходимо разрабатывать варианты представления знаний, менее зависящие от кода и позволяющие пользователям описывать предметные области в более понятном им виде. Современные интеллектуальные системы используют в качестве источников информации базы знаний вместо баз данных. Основными преимуществами использования баз знаний являются:

- возможность хранить сложные разнородные сведения;
- возможность расширять и дополнять описание предметной области без перепрограммирования;
- наглядность и доступность представления знаний пользователю.

На данный момент для представления знаний в сложных информационных системах широко используется ряд методов: семантические сети [1], фреймы [2], формальные логические модели [3], продукционные модели [4]. Каждый из этих методов обладает своими преимуществами и недостатками, а также ограничениями на описываемые знания, предметную область и системы, в которых он может использоваться.

Онтологический подход к представлению и хранению знаний позволяет с помощью онтологий описать любую разнородную, сколь угодно сложную предметную область. С точки зрения разработки интеллектуальных систем, используя этот подход, можно формализовать специфические предметные знания в виде, допускающем компьютерную обработку, и отделить знания от программного кода системы. Возможность для пользователей онтологии добавлять и изменять объекты, атрибуты и классы по мере уточнения целей и задач делает этот метод оптимальным в практическом использовании.

Разделы №1–№2 посвящены рассмотрению основных понятий и определений, классификации онтологий по различным критериям, описанию онтологий верхнего уровня.

В Разделах №3–№4 описываются основные понятия семантической сети, стек Semantic Web, архитектура метаданных WWW, модель представления данных, языки представления онтологий

В Разделах №5–№6 описывается понятие модели онтологии, онтологические базисы, методы и средства представления онтологий, а также программные инструменты для работы с онтологией.

В заключении обсуждаются перспективы развития данного направления, и показывается возможный облик будущих систем, появление которых ожидается уже в самое ближайшее время.

Учебное пособие, в первую очередь, предназначено для обучающихся по направлению подготовки 09.04.01 Информатика и вычислительная техника. Учебное пособие может быть полезно широкому кругу читателей, включая инженеров и специалистов, ученых, аспирантов и студентов в самых различных сферах деятельности, а также всем тем, кто интересуется перспективами развития новых информационных технологий.

1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ ОНТОЛОГИЙ. КЛАССИФИКАЦИЯ ОНТОЛОГИЙ

Онтология – это формализованные концептуальные знания о предметной области, представленные в форме, допускающей компьютерную обработку и используемые в правилах принятия решений. Концептуальность знаний онтологии означает, что эти знания формулируются в терминах основных понятий и отношений, описывающих фрагменты окружающего мира.

Онтологии используются в качестве основы базы знаний об основных понятиях предметной области и связях между ними, этим обеспечивается единство терминологии и предоставляется возможность расширения знаний системы в случае появления новых концептов и связей, а также появляется возможность формально специфицировать ситуации в реальности.

1.1. Определение онтологии

Существует множество определений онтологии, наиболее широко используемым является следующее: ***«Онтология – это явная спецификация концептуализации»*** [5]. Здесь *концептуализация* означает абстрактное представление предметной области.

Выделим основные понятия этого определения онтологии:

- онтология определяет (специфицирует) концепты, отношения и другие сущности, предназначенные для моделирования предметной области;
- спецификация принимает форму лексических определений (классов, отношений и т.д.), которые обеспечивают значе-

ия для словаря и формальные ограничения на их использование.

Данное определение представляется слишком широким – от простых глоссариев до логических теорий в исчислениях предикатов. Но это справедливо для моделей данных любой сложности, например, реляционная база данных из одной таблицы по-прежнему является экземпляром реляционной модели данных. Онтология является инструментом инженерии знаний, что определяет ее использование. Из этого следует, что онтологии обеспечивают средства, позволяющие создавать экземпляры модели предметной области в базах знаний, формировать запросы на предоставление услуг, основанных на знаниях, и результатов вызова таких услуг.

Например, поисковая служба может предложить только текстовый глоссарий терминов для формулирования запросов, который будет выступать в качестве онтологии. С другой стороны, стандарт W3C Semantic Web содержит определенный язык для кодирования онтологий (OWL) в нескольких форматах, которые различаются по выразительной мощности. Это означает, что онтология является спецификацией абстрактной модели данных, не зависящей от конкретной формы ее представления (*концептуализация* предметной области).

Распространено также определение: ***«Онтология – общее понимание некоторой области интереса»*** [6]. В контексте компьютерных и информационных наук, онтология определяет набор примитивов представления, с помощью которых можно моделировать область знаний. Примитивы представления, как правило, включают классы (наборы), атрибуты (свойства) и отношения (отношения между членами класса) [7]. Определения примитивов включают информацию об их значениях и ограничения на их логическую непротиворечивость. В контексте систем баз данных онтология может рассматриваться как уровень абстракции моделей данных, анало-

гичных иерархическим и реляционным моделям, но предназначенных для моделирования знаний о сущностях, их атрибутах, их отношениях с другими сущностями.

Онтологии, как правило, описываются на языках, обеспечивающих абстрагирование от структур данных и стратегии реализации; на практике, языки онтологий по выразительной силе ближе к логике предикатов первого порядка, чем языки, используемые для моделирования баз данных. По этой причине говорят, что онтологии определены на «семантическом» уровне, а схемы баз данных являются моделями данных на «логическом» или «физическом» уровнях. Благодаря своей независимости от моделей данных более низкого уровня, онтологии используются для интеграции разнородных баз данных за счет взаимодействия несовместимых систем и определения интерфейсов независимых сервисов, основанных на знаниях. В технологии стандартов стека семантического веба [8] онтологии выделяются как отдельный слой. В настоящее время существуют стандартные языки, а также множество коммерческих и открытых инструментов для создания онтологий и работы с ними.

1.1.1. Исторические основы

Термин «онтология» происходит из области философии, связанной с изучением бытия или существования. Онтология в философском смысле – учение о бытии, о сущем, система категорий, являющихся следствием определенного взгляда на мир. В философии говорят об онтологии как о теории природы существования (например, онтология Аристотеля определяет базовые категории вещество и качество, с помощью которых предлагается описывать все, что существует). В компьютерных и информационных науках онтология – это технический термин, обозначающий способ представления знаний о фрагменте окружающего мира, т.е. знания «как они есть»

(«knowledge as it is»), способ моделирования знаний о некоторой предметной области, реальной или виртуальной [7].

Термин «онтология» был принят исследователями в области искусственного интеллекта (ИИ), полагавшими, что новые онтологии можно создавать как вычислительные модели, позволяющие выполнять определенные виды автоматизированных рассуждений на основе математической логики. В1980-х годах сообщество ИИ начало использовать термин «онтология» для наименования теории моделируемого мира и компонентов систем знаний. Некоторые исследователи, черпая вдохновение из философских онтологий, рассматривали вычислительную онтологию как вид прикладной философии.

В начале 1990-х годов усилия по созданию стандартов совместимости определили набор технологий, в котором слой онтологии рассматривался как стандартный компонент систем знаний, а понятие онтологии стало употребляться как технический термин в информатике. Онтология определяется как «явная спецификация концептуализации», содержащая объекты, концепты и другие сущности, которые, как предполагается, существуют в некоторой области интересов и отношений, которые определены между ними.

1.1.2. Научные основы

Онтология, обсуждаемая здесь, в контексте инжиниринга программного обеспечения и баз данных, имеет также теоретическое обоснование [8]. Онтология определяет словарь для формирования утверждений, которые могут быть входами или выходами агентов знаний (как в программе).

В качестве спецификации интерфейса онтология обеспечивает язык для общения с агентами. Агенту, поддерживающему этот интерфейс, не требуется использовать термины онтологии

для внутреннего кодирования его знаний. Тем не менее, определения и формальные ограничения онтологии должны накладывать ограничения на то, что может быть осмысленно описано на этом языке. Поддержка интерфейса с помощью онтологического словаря требует, чтобы входные и выходные утверждения логически соответствовали определениям и ограничениям онтологии. Это аналогично требованию, чтобы строки таблицы базы данных (или утверждения в SQL) были совместимы с ограничениями целостности, которые заданы декларативно и независимо от внутренних форматов данных.

Аналогично, в то время как онтологию требуется сформулировать на каком-то языке представления, она должна быть спецификацией семантического уровня, т.е. не должна зависеть от стратегии моделирования данных или реализации. Например, обычная модель базы данных может представлять идентификатор экземпляра, использующего первичный ключ, который присваивает уникальный идентификатор каждому экземпляру. Однако, идентификатор первичного ключа является артефактом процесса моделирования и ничего не обозначает в предметной области. Онтологии, как правило, формулируются на языках, которые по выразительной силе ближе к логическим формализмам, таким как исчисления предикатов. Это позволяет разработчику онтологий задавать семантические ограничения независимо от стратегии кодирования.

Если онтологии кодируются с использованием стандартных формализмов, можно повторно использовать большие, ранее разработанные онтологии. В этом контексте онтологии воплощают результаты научных исследований, а также предлагают оперативный метод сопоставления теории практике в системах баз данных.

1.1.3. Основные области применения

Онтологии являются частью стандартов стека W3C для Semantic Web, в котором они используются для определения стандартных концептуальных словарей, обеспечивающих обмен данными между системами за счет предоставления услуг ответов на запросы, повторного использования баз знаний, взаимодействия между несколькими разнородными системами и базами данных.

Основная роль онтологий в отношении систем управления базами данных – представление модели данных на более высоком уровне абстракции, чем конкретные конструкции баз данных (логическом или физическом), так что данные могут быть экспортированы, переведены, запрошены и унифицированы между самостоятельно разработанными системами и услугами.

Онтологии успешно применяются для взаимодействия баз данных, перекрестного поиска данных в базах и интеграции веб-сервисов [7].

1.2. Классификация онтологий

При рассмотрении онтологий условно можно выделить два направления, до некоторого времени развивавшихся отдельно [9]. Первое связано с представлением онтологии как формальной системы, основанной на математически точных аксиомах. Второе направление развивалось в рамках компьютерной лингвистики и когнитивной науки. Там онтология понималась как система абстрактных понятий, существующих только в сознании человека, которая может быть выражена на естественном языке (или средствами какой-то другой системы символов). При этом обычно не делается предположений о точности или непротиворечивости такой системы.

Таким образом, существует два альтернативных подхода к созданию и исследованию онтологий. Первый (формальный) основан на логике (предикатов первого порядка, дескриптивной, модальной и т. п.). Второй (лингвистический) основан на изучении естественного языка (в частности, семантики) и построении онтологий на больших текстовых массивах, так называемых корпусах.

В настоящее время данные подходы тесно взаимодействуют. Идет поиск связей, позволяющих комбинировать соответствующие методы. Поэтому иногда бывает сложно отделить лексические онтологии с элементами формальных аксиоматик от логических систем с включениями лингвистических знаний.

Независимо от различных подходов можно выделить следующие основные принципы классификации онтологий:

- по степени формальности;
- по наполнению, содержимому.

1.2.1. Классификация по степени формальности. «Спектр онтологий»

Обычно люди и компьютерные агенты (программы) имеют некоторое представление о значениях терминов. Программные агенты иногда предоставляют спецификацию входных и выходных данных, которые также могут быть использованы как спецификация программы. Сходным образом онтологии могут быть применены, чтобы предоставить конкретную спецификацию имен терминов и значений терминов. В рамках этого понимания (где онтология является спецификацией концептуальной модели – концептуализации) существует простор для вариаций. Отдельные виды онтологий могут быть представлены как точки на спектре в зависимости от деталей их реализации.

На рис. 1 изображен так называемый спектр онтологий по степени формальности представления, использованию тех или иных

синтаксических конструкций (косая черта разделяет системы, представляющие «машино-понятные» и «человеко-понятные» описания). Каждая точка соответствует наличию некоторых ключевых структур в онтологии, отличающих ее от других точек на спектре. Косая черта условно отделяет онтологии от других ресурсов, имеющих онтологический характер.

Рис. 1. Спектр онтологий

Первой точке на спектре соответствует контролируемый словарь, т.е. конечный список терминов (простейшим примером является каталог на основе идентификаторов). Каталоги представляют точную (не многозначную) интерпретацию терминов. Например,

каждый раз, ссылаясь на термин «машина», мы будем использовать одно и то же значение (соответствующее некоторому ID в словаре), вне зависимости от того, о чем идет речь в контексте: о «стиральной машине», «автомобиле» или «государственной машине».

Другой спецификацией онтологии может быть глоссарий, представляющий список терминов с их значениями. Значения описываются в виде комментариев на естественном языке. Это дает больше информации, поскольку люди могут прочесть такой комментарий и понять смысл термина. Интерпретации терминов могут быть многозначными. Глоссарии непригодны для автоматической обработки программными агентами, но можно, как и ранее, присвоить терминам ID.

Тезаурусы несут дополнительную семантику, определяя связи между терминами. Отношения, свойственные для тезаурусов: синонимия, иерархическое отношение и ассоциация. Ранние иерархии терминов, появившиеся в Сети, определяли термины через операции обобщения и уточнения. Yahoo, например, ввела небольшое число категорий верхнего уровня, таких, как «предметы одежды». Затем «Платье» определялось как вид (женской) одежды. Явная иерархия Yahoo не соответствовала в точности формальным свойствам иерархического отношения isA. В таких иерархиях может встретиться ситуация, в которой экземпляр класса-потомка не является экземпляром класса-предка. Например, общая категория «предметы одежды» имеет подкатегорию «женские» (которая должна была бы более точно называться «женские предметы одежды»), а эта категория, в свою очередь, включает подкатегории «аксессуары» и «платья». Ясно, что аксессуары, например, «броши», не являются предметами одежды. Здесь не выполняется важное свойство отношения isA – транзитивность.

Далее следует точка «формальные таксономии». Эта разновидность онтологий включает точное определение отношения isA

(«КЛАСС-ПОДКЛАСС»). В таких системах строго соблюдается транзитивность отношения isA: если B является подклассом класса A, то каждый подкласс класса B также является подклассом класса A. А для отношения «КЛАСС-ЭКЗЕМПЛЯР» (isInstanceOf) выполняется следующее свойство: если B является подклассом класса A, то каждый экземпляр класса B также является экземпляром класса A. Поэтому в приведенном выше примере «броши» не могут быть помещены ниже в иерархии «предмет одежды», даже в подкатегорию «женские», или стать экземпляром этой категории. Строгая иерархия необходима при использовании наследования для процедуры логического вывода.

Следующая точка – наличие формального отношения «КЛАСС-ЭКЗЕМПЛЯР». Некоторые классификации включают только имена классов, другие содержат на нижнем уровне экземпляры (индивиды). Данная точка спектра допускает наличие у классов экземпляров (примеров).

Далее среди структурных элементов появляются слоты. Здесь классы (иногда их называют фреймами) могут иметь информацию о свойствах (слотах). Например, класс «ПРЕДМЕТ ОДЕЖДЫ» может иметь свойства «цена», «сделанИз». Свойства бывают особенно полезными, когда они определены на верхних уровнях иерархии и наследуются подклассами. В потребительской иерархии класс «ПРОДУКТ» может иметь свойство «цена», которое получат все его подклассы.

Большей выразительностью обладают онтологии, включающие ограничения на область значений свойств. Значения свойств берутся из некоторого предопределенного множества (целые числа, символьные константы) или из подмножества концептов онтологии (множество экземпляров данного класса, множество классов). Можно ввести дополнительные ограничения на то, что может заполнять свойство. Например, для свойства «сделанИз» класса

«предмет одежды» значения могут быть ограничены экземплярами класса «Материал». Легко увидеть проблемы, которые могут возникнуть в этом случае при использовании нестрогой таксономии. Если «духи» – потомок класса «предмет одежды», он унаследует свойство «сделанИз» вместе с ограничением («Материал»).

В целом с необходимостью описывать более сложные факты выразительные средства онтологии (и ее структура) усложняются. Например, может потребоваться заполнить значение какого-либо свойства экземпляра, используя математическое выражение, основанное на значениях других свойств данного экземпляра или значениях свойств других экземпляров. Многие онтологии позволяют объявлять два и более класса дизъюнктивными (непересекающимися). Это означает, что у данных классов не существует общих экземпляров.

Некоторые языки описания онтологий позволяют делать произвольные логические утверждения о концептах – аксиомы.

Языки описания онтологий, подобные CycL и Ontolingua, позволяют фиксировать утверждения на языке логики предикатов первого порядка (FOL).

1.2.2. Классификация по цели создания

В рамках этой классификации выделяют 4 уровня (рис. 2): онтологии представления, онтологии верхнего уровня, онтологии предметных областей и прикладные онтологии [9].

1.2.2.1. Онтологии представления

Цель создания онтологий представления: описать область представления знаний, создать язык для спецификации других онтологий более низких уровней. Пример: описание понятий языка OWL средствами RDF/RDFS (рис. 3). В данном описании определяются такие понятия, как «класс», «отношение», «ограничение на значение свойства», «домен», «диапазон» и т.п.

Рис. 2. Классификация онтологий по цели создания

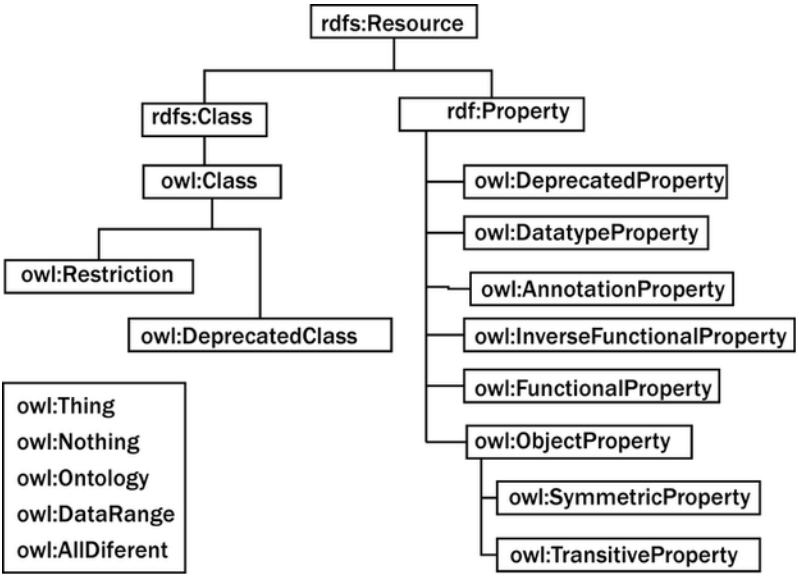

Рис. 3. Онтология представления для языка OWL

1.2.2.2. Онтологии верхнего уровня

Назначение онтологий верхнего уровня – в создании единой онтологии, фиксирующей знания, общие для нескольких предметных областей, и многократном использовании данной онтологии. Существует несколько крупных проектов: SUMO, Sowa's Ontology, Cyc. Но в целом попытки создать онтологию верхнего уровня на все случаи жизни пока не привели к ожидаемым результатам. Многие онтологии верхнего уровня похожи друг на друга. Они содержат одни и те же концепты: Сущность, Явление, Процесс, Объект, Роль и т.п. (рис. 4).

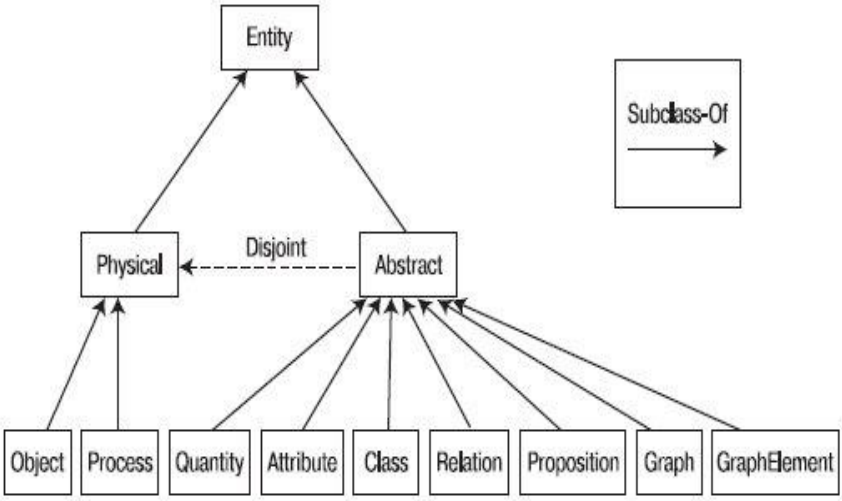

Рис. 4. Верхний уровень иерархии онтологии SUMO

1.2.2.3. Онтологии предметных областей

Назначение онтологий предметных областей заключается в обобщении понятий, которые повторно используются внутри одной предметной области.

1.2.2.4. Прикладные онтологии

Назначение прикладных онтологий в том, чтобы описать концептуальную модель конкретной задачи или приложения. Они содержат наиболее специфичную информацию. Примеры проектов: TOVE, Plinius.

2. ОНТОЛОГИИ ВЕРХНЕГО УРОВНЯ

Под онтологией верхнего уровня понимают систему, которая состоит из множества понятий, их определений и аксиом, необходимых для ограничения интерпретации и использования понятий [9]. При решении прикладных задач онтология часто отождествляется с набором классов (понятий предметной области), связанных определенным набором отношений. Базовыми типами отношений являются «ПОДКЛАСС-НАДКЛАСС» (гипонимия), «ЧАСТЬ-ЦЕЛОЕ» (меронимия), «ЭКЗЕМПЛЯР-КЛАСС», «ПРИЧИНА-СЛЕДСТВИЕ», отношение зависимости и др.

В основном онтологии являются согласованными или разделяемыми (shared) ресурсами: содержимое онтологии может одновременно использоваться несколькими лицами, группами или сообществами. Онтологии верхнего уровня содержат знания здравого смысла (common sense) о моделируемом мире, формируя единую для онтологий нижних уровней систему понятий.

Рассмотрим и сравним наиболее масштабные проекты онтологий верхнего уровня [9].

2.1. OpenCyc

OpenCyc – открытая для общего пользования часть коммерческого проекта Cyc, в рамках которого создана наиболее масштабная и детализированная на текущий момент онтология в области здравого смысла. База знаний OpenCyc содержит информацию из различных предметных областей: Философия, Математика, Химия, Биология, Психология, Лингвистика и т.д. Файл с описаниями

OpenCyc имеет объем около 700 мегабайт и доступен для скачивания с сайта проекта [10].

Ключевым понятием в проекте OpenCyc является коллекция (рис. 5). Любая коллекция может содержать подколлекции и экземпляры. Таким образом, в OpenCyc определены два таксономических отношения: «ПОДКОЛЛЕКЦИЯ-НАДКОЛЛЕКЦИЯ» (genls) и «ЭКЗЕМПЛЯР-КОЛЛЕКЦИЯ» (isA). Экземпляром коллекции может быть любой термин онтологии. Важная черта отношения isA в том, что оно передается по иерархии отношения genls, т.е. если A является экземпляром коллекции B и B является подколлекцией коллекции C, то A является также экземпляром коллекции C. В случае, если коллекции A и B связаны отношением genls (A genls B), это означает, что все экземпляры коллекции A являются также экземплярами коллекции B.

Рис. 5. Фрагмент иерархии коллекций в OpenCyc

В вершине иерархии коллекций находится универсальная коллекция с именем «Нечто». По определению, она содержит все, что существует в рамках описываемой области (т.н. «Universe of

Discourse»). Любая коллекция, описанная в OpenCyc, будь то «Индивид», «МатематическоеМножество» или «Коллекция», является и подколлекцией, и экземпляром коллекции «Нечто». Более того, коллекция «Нечто» является как экземпляром, так и подколлецией самой себя, но не подколлекцией какой-либо другой коллекции. На первом уровне иерархии «Нечто» разделяется сразу на 116 подколлекций. На рис. 5 изображена урезанная иерархия коллекций верхних уровней.

Коллекция «Индивид» содержит всевозможные индивиды, т. е. сущности, не являющиеся ни множествами, ни коллекциями. Индивиды могут быть абстрактными или конкретными, описывать физические объекты, события, отношения, числа, группы, они могут состоять из частей, иметь сложную структуру, но ни один экземпляр этой коллекции не может иметь элементов или подмножеств. Так, индивид, имеющий части (связи типа «ЧАСТЬ-ЦЕЛОЕ»), и множество или коллекция, содержащая те же самые части (связи типа «ЭЛЕМЕНТ-МНОЖЕСТВО» и «ЭЛЕМЕНТ-КОЛЛЕКЦИЯ», – две совершенно разные сущности. Например, данная фирма (1), группа, содержащая всех работников данной фирмы (2), коллекция всех работников фирмы (3) и множество всех работников фирмы (4) – четыре разных понятия и только первые два из них – индивиды.

Коллекция «Коллекция» содержит все коллекции онтологии OpenCyc, кроме «Нечто». Именно «Коллекция» наиболее близка понятию класс, которое часто используют при проектировании онтологий предметных областей (но не понятию класс объектно-ориентированного программирования!), поскольку эта коллекция описывает набор объектов (экземпляров коллекции), имеющих некоторые общие атрибуты (свойства). Это же отличает «Коллекцию» от «МатематическогоМножества». Множество может содержать абсолютно не связанные элементы, а «Коллекция» – нет. Все экземпляры «Коллекции» являются абстрактными сущностями, даже если коллекция содержит материальные объекты.

Структурно база знаний OpenCyc состоит из констант (терминов) и правил (формул), оперирующих этими константами. Правила делятся на два вида: выводимые утверждения и аксиомы. Под аксиомами в OpenCyc понимаются утверждения, которые были явно и вручную введены в базу знаний экспертами, а не появились там (или могут появиться) в результате работы машины вывода. Все утверждения или формулы в базе знаний OpenCyc фиксируются на языке CycL, выразительно эквивалентном исчислению предикатов первого порядка.

2.2. DOLCE

DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering) – первая из онтологий в библиотеке базовых онтологий проекта WonderWeb [11].

Онтологию DOLCE предполагается применять в SemanticWeb для согласования между интеллектуальными агентами, использующими разную терминологию. При этом онтология не претендует на звание универсальной, стандартной или общей. Основная цель разработчиков – создать модель, помогающую при сравнении и объяснении связей с другими онтологиями библиотеки WFOL (базовой библиотеки онтологий WonderWeb), а также для выявления скрытых допущений, лежащих в основе существующих онтологий и лингвистических ресурсов, таких как WordNet. DOLCE имеет когнитивный уклон, поскольку фиксирует онтологические категории естественного языка и знания «здравого смысла».

В основу процесса проектирования легло фундаментальное философское разделение всех сущностей на универсалии (сущности, потенциально или реально имеющие экземпляры) и индивиды (или частности), которые не имеют и не могут иметь экземпляров. DOLCE – онтология индивидов, в том смысле, что область описа-

ния ограничена только ими. В качестве примера универсалии можно привести понятие «Собака» (оно имеет множество экземпляров, конкретных примеров в окружающем мире). В отличие от этого понятия, понятие «Время» скорее рассматривается как индивид (едва ли кому-то понадобится трактовать «Время» как множество различных сущностей, конечно, если речь не идет о параллельных мирах).

Еще одна черта DOLCE (также заимствованная разработчиками из философии) – явное разделение на «Постоянные» и «Происходящие» сущности. Различие между ними состоит в том, что «Постоянные» сущности имеются в наличии целиком и неизменно в некотором фиксированном промежутке времени (например, стол или дом в течение периода своего существования). Верхние уровни иерархии онтологии DOLCE представлены на рис. 6.

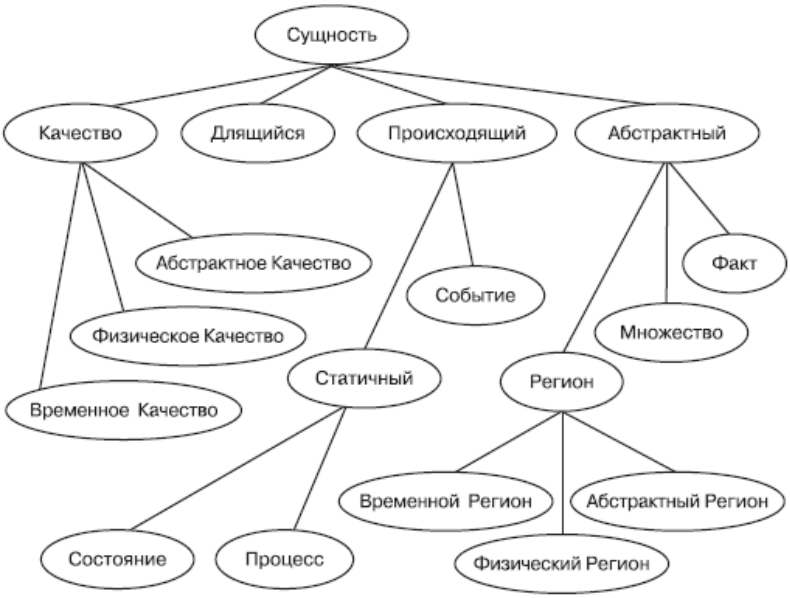

Рис. 6. Верхние уровни иерархии DOLCE

«Происходящие» разворачиваются во времени и в каждый момент в некотором временном интервале они могут быть различными, по-разному себя проявлять, иметь разный состав (например, ураган или период раннего Ренессанса), однако при этом их идентичность сохраняется.

Такое разделение на «объект» и «процесс» весьма условно. В онтологии определены два типа отношения «ЧАСТЬ-ЦЕЛОЕ». Первое никак не зависит от времени, второе имеет временной индекс, определяющий, в каких временных рамках отношение действует. Подобное «раздвоение» наблюдается и для отношения «КАЧЕСТВО-ОБЛАДАТЕЛЬ КАЧЕСТВА». Другие базовые отношения онтологии: «УЧАСТНИК-ПРОЦЕСС», «КОМПОНЕНТ-ЦЕЛОЕ» (компонент входит в состав целого) и отношение зависимости имеют временной индекс. Для сравнения, в онтологии OpenCyc нет явного деления на «Постоянные» и «Происходящие». Поэтому среди множества отношений в разделе «Части объектов» нет отношения, учитывающего временной аспект: возможное непостоянство данного отношения.

Для представления своей онтологии авторы DOLCE избрали более гибкий, чем в проекте Cyc, подход: онтология фиксируется с использованием логики предикатов первого порядка. Затем описывается та часть утверждений, которая может быть представлена на языке OWL. Оставшиеся аксиомы, выраженные на языке KIF, добавляются к OWL-описаниям в виде комментариев. Таким образом, достигается выразительность уровня KIF и совместимость с OWL.

2.3. SUMO

SUMO (Standard Upper Merged Ontology) – онтология верхнего уровня, разработанная в рамках проекта рабочей группы

IEEE SUO (IEEE Standard Upper Ontology Working Group) и Teknowledge. Проект претендует на статус стандарта для онтологий верхнего уровня [12].

Онтология SUMO содержит наиболее общие и самые абстрактные концепты, имеет исчерпывающую иерархию фундаментальных понятий (около 1 тыс.), а также набор аксиом (примерно 4 тыс.), определяющих эти понятия. Назначение SUMO – содействовать улучшению интероперабельности данных, извлечения и поиска информации, автоматического вывода и обработки естественного языка. Онтология охватывает следующие области знания: общие виды процессов и объектов, абстракции (теория множеств, атрибуты, отношения), числа и единицы измерения, временные понятия, части и целое, агенты и намерения. SUMO является «канонической» онтологией верхнего уровня: содержит обозримое число концептов и аксиом, имеет ясную иерархию классов, легко расширяется, является итогом объединения различных общедоступных онтологий верхнего уровня. К преимуществам SUMO можно отнести возможность трансляции описания онтологии на любой из основных языков представления знаний, наличие онтологии среднего уровня (MILO), гладко интегрированной с SUMO, несколько дюжин примеров практического применения, а также связь с WordNet – наиболее крупным на настоящий момент тезаурусом, содержащим около 150 тыс. слов английского языка.

Иерархия классов онтологии SUMO показана на рис. 7.

Иерархия классов в SUMO более прозрачна, чем в OpenCyc, и, возможно, более удобна для практического применения, чем DOLCE. Основными концептами, как во многих онтологиях верхнего уровня, являются «Сущность» и ее категории – «Физический» и «Абстрактный». Первая категория включает все, что имеет положение в пространстве-времени, а вторая – все остальное (а точнее только то, что существует в сознании). «Физический» делится на

«Объект» и «Процесс», что соответствует подходу, реализованному в DOLCE. Непосредственно под концептом «Объект» находятся два непересекающихся понятия: «СвязныйОбъект» и «Коллекция». Первое обозначает любой объект, все части которого непосредственно или косвенно связаны друг с другом. Концепт «СвязныйОбъект» разделен на два концепта: «НепрерывныйОбъект» и «ДискретныйОбъект». «НепрерывныйОбъект» характеризуется тем, что все его части (вплоть до самого низкого уровня деления) имеют такие же свойства, как и целое. Такие субстанции, как вода и глина, могут быть подклассами концепта «НепрерывныйОбъект», так же, как поверхности и географические области.

Рис. 7. Иерархия классов SUMO

«Коллекции» в SUMO отделены от «СвязныхОбъектов». «Коллекции» строятся из несвязанных частей и с использованием отношения «ЧЛЕН-КОЛЛЕКЦИЯ» между частями и соответствующей им коллекцией. Здесь, так же как в OpenCyc, проводится разграничение понятий «Коллекция», «Класс» и «Множество». Предикат «быть членом коллекции» отличен от предикатов «быть

экземпляром класса» и «быть элементом множества», относящих объекты к понятиям «Класс» или «Множество», которым они соответствуют. В отличие от «Классов» и «Множеств», «Коллекции» занимают некоторое положение в пространстве-времени (они не абстрактны, как в OpenCyc, а материальны), члены могут добавляться и удаляться из коллекции, не меняя ее идентичности. Примеры «Коллекций»: ящики с инструментами, футбольные команды.

Возвращаясь к концептам уровня «Физический»-«Абстрактный», обсудим ветвь «Абстрактный». Категория «Абстрактный» разделяется на «Множество», «Утверждение», «Величина» и «Атрибут». «Множество» – обычное понятие теории множеств, включает «Класс», который, в свою очередь, имеет подкласс «Отношение». «Класс» понимается как множество со свойством или пересечением свойств, которые определяют принадлежность к «Классу», «Отношение» есть «Класс» упорядоченных пар. «Отношение» по смыслу ближе к «Классу», чем к «Множеству». «Отношение» ограничено только теми упорядоченными парами, которые описывают его содержимое.

Концепт «Утверждение» соответствует понятию семантического или информационного содержимого. Однако SUMO не накладывает никаких ограничений на это содержимое. Это более общее понятие, чем используемое в большинстве онтологий; почти невозможно принципиально разделить абстрактное содержимое, выраженное одним предложением, и абстрактное содержимое, выраженное многочисленными речевыми единицами. Примеры «Утверждений»: краткое изложение рассказа, музыкальное содержимое напечатанной партитуры.

Понятие «Атрибут» включает все количества, свойства и т. д., которые не представимы как «Объекты». Например, вместо того чтобы делить класс «Животные» на «ЖивотныеЖенскогоПола» и «ЖивотныеМужскогоПола», создаются экземпляры «Женский» и

«Мужской» класса «БиологическийАтрибут», который является подклассом «Атрибута».

Наконец, «Величина» разделяется на «Число» и «ФизическаяВеличина». Первое понимается как независящая от системы измерения величина, а второе – как составная величина, состоящая из «Числа» и конкретной единицы измерения.

Аксиомы ограничивают интерпретацию концептов и предоставляют основу для систем автоматизированного рассуждения, которые могут обрабатывать базы знаний, соответствующие по своей структуре онтологии SUMO. Пример аксиомы: «Если C является экземпляром процесса горения, то существуют выделение тепла H и излучение света L такие, что оба они – H и L – являются подпроцессами C». Более сложные предложения говорят, что процессы выделения тепла и излучения света сопутствуют каждому процессу горения. Аксиомы кодируются в SUMO на формальном логическом языке SUO-KIF.

2.4. Области применения онтологий предметных областей и прикладных онтологий

Создание и дальнейшее управление сложными информационными комплексами – задача, требующая наличия эффективной, гибкой модели представления знаний, способной к расширению, реконфигурации и развитию с учетом потребностей использования.

Использование онтологий и семантических технологий для решения данной задачи обеспечивает следующие основные преимущества:

- единый базис, в котором описываются знания, что позволяет формализовать практически любую предметную область для использования системой без необходимости программных доработок;

- легкость расширения и внесения изменений в структуру представления знаний – онтология позволяет легко и безопасно (сохраняя согласованность знаний) вносить изменения в существующие знания;
- получение логических выводов на основании легко расширяемого набора правил и аксиом, с целью автоматического связывания информации, валидации, поиска дубликатов и т.д.;
- обеспечение доступности для восприятия пользователями больших объемов сложно структурированной информации;
- возможность выполнения имитационного моделирования процессов с целью их оптимизации;
- решение ряда технических задач, прежде всего в области интеграции информационных систем.

Рассмотрим основные широко распространенные задачи, успешно решаемые с помощью применения онтологий:

1. Проведение эффективного поиска в различных хранилищах данных, к которым можно отнести как базы данных, каталоги, статьи, так и базы знаний. Среди перспектив развития данной области можно выделить формирование запросов на естественном языке. На данный момент это организовано в большом количестве поисковиков, но многие из них не справляются со своей задачей, принося в ответы по запросу большое количество информационного шума. Однако стоит отметить, что прогресс, осуществленный в данном направлении, существенен. Ряд семантических поисковиков, таких как AskNet или hakia способны отвечать на некоторые простые вопросы.

2. Разработка систем, реализующих технологию рассуждений. К таким системам можно отнести экспертные системы, системы управления. Для сложных по своей структуре систем актуальны проблемы, связанные непосредственно с человеческим фактором.

Программы, разработанные на основе технологии рассуждений, имеют блок объяснения решения, в котором должны раскрывать суть протекающих процессов и их причины, а также приводить доводы к предлагаемым вариантам решений. Наличие такого модуля дает возможность действовать осмысленно, т.к. при принятии решений и при объяснении должна учитываться семантика как отдельных терминов, так и составленных из них высказываний и их композиций. Именно использование онтологий способствует достижению данной цели.

3. Поиск по смыслу текста. Смысловой поиск не может быть осуществлен без методов и алгоритмов извлечения семантики как самой текстовой документации, так и составляемого запроса. Для организации поиска по смыслу в текстовой информации необходимы методы извлечения семантики из текстовых документов и запросов и сопоставления получаемых семантических представлений. Данные алгоритмы смогут обеспечить рост эффективности автоматического извлечения из текста основного содержания – реферирования, аннотирования и классификации документов.

4. Семантический поиск в сети Интернет. Сложная, неупорядоченная структура сети Интернет затрудняет эффективное извлечение данных. Онтологии дают возможность организации информационных профилей узлов сети. Такой подход позволяет не только уменьшить время на поиск, но и сократить нагрузку на сеть.

5. Создание общей терминологии. Суть Semantic Web состоит в автоматизации «интеллектуальных» задач обработки значения (в семантическом смысле) тех или иных ресурсов, имеющихся в Интернет. Обработкой информации и ее обменом должны заниматься не люди, а специальные интеллектуальные агенты (программы, размещенные в Интернет). Но для того, чтобы взаимодействовать между собой, агенты должны иметь общее (разделяемое всеми) формальное представление значения для любого ресурса. Именно

для цели представления общей, явной и формальной спецификации значения в Semantic Web используются онтологии.

2.5. Контрольные вопросы по разделам №1-№2

1. Приведите определения онтологии. Поясните смысл определений.
2. Опишите исторический аспект понятия «онтология».
3. Из чего состоит онтология? Что она определяет?
4. Перечислите основные области применения онтологии.
5. По каким признакам выполняется классификация онтологий?
6. Что такое «спектр онтологий»? Какому признаку классификации онтологий он соответствует? Какие компоненты он содержит?
7. Перечислите уровни в рамках классификации онтологий по целям создания.
8. Каково назначение и особенности реализации онтологии представления?
9. Каково назначение и особенности реализации онтологии верхнего уровня?
10. Какие онтологии верхнего уровня Вам известны?
11. Каково назначение и особенности реализации онтологий предметных областей и прикладных онтологий?

3. SEMANTIC WEB

3.1. Понятие семантической сети

Идея Семантической Сети (Semantic Web) впервые была провозглашена в 2001 году Тимом Бернерсом-Ли (создателем World Wide Web) [7]. Суть ее состоит в автоматизации «интеллектуальных» задач обработки значения (в семантическом смысле) тех или иных ресурсов, имеющихся в Сети. Обработкой и обменом информацией должны заниматься не люди, а специальные интеллектуальные агенты (программы, размещенные в Сети). Но для того, чтобы взаимодействовать между собой, агенты должны иметь общее (разделяемое всеми) формальное представление значения для любого ресурса. Именно для цели представления общей, явной и формальной спецификации значения в Semantic Web используются онтологии.

«Semantic Web – это расширение Web, в котором информации придаётся определённая семантика, позволяя людям и машинам работать вместе» – примерно такое определение дают своим работам члены W3C Semantic Web Activity. Целью этого проекта является внедрение в Web таких технологий, которые позволят существенно повысить уровень интеграции информации, обеспечить развитую машинную обработку данных, дадут возможность выдавать более адекватные ответы на поисковые запросы и т.д.

Текущее состояние Web характеризуется слабой структурированностью данных, низким уровнем их взаимосвязи. Распространение XML-технологий дает возможность структурировать информацию, обеспечить синтаксическую интероперабельность

приложений. Semantic Web является логическим продолжением развития Web – от гипертекстовых страниц к XML-данным, а от XML – к смысловому содержанию и объединению разбросанной в Web информации.

3.2. Стек Semantic Web

На рис. 8 представлена диаграмма, называемая стеком Semantic Web.

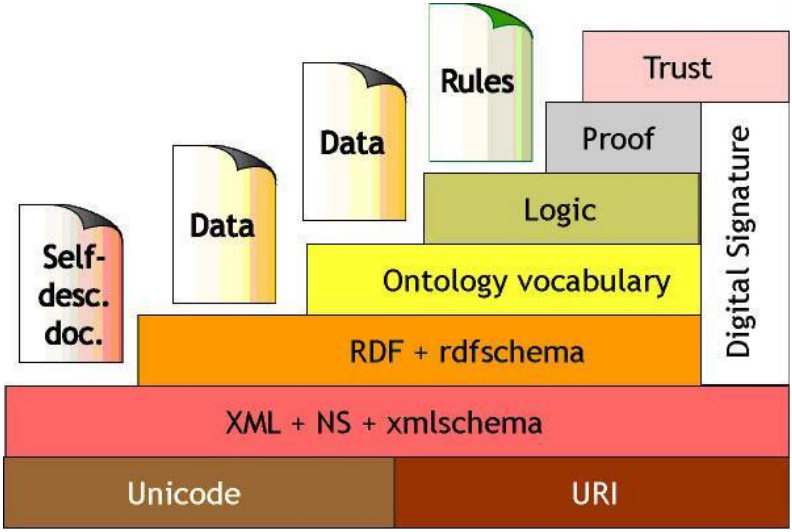

Рис. 8. Стек Semantic Web

Semantic Web базируется на модели данных Resource Description Framework (RDF), которая позволяет объединить информацию из различных источников, включая базы данных и системы инженерии знаний. RDF может быть наиболее полезен в обеспечении совместного использования информации, смысл которой может одинаково интерпретироваться различными программными

агентами. Второй базовый компонент Semantic Web – это RDF/XML-синтаксис, который позволяет представить RDF-данные в XML-виде. Следующий уровень в пирамиде технологий Semantic Web занимает язык RDF Schema – язык описания словарей RDF-терминов (классов и свойств Web-ресурсов). RDFS служит фундаментом для более богатых языков описания онтологий предметной области, которые позволяют адаптировать к Web системы логики и обеспечить семантическую обработку данных.

Работа над средствами описания семантики в Сети началась задолго до публикации 2001 года. В 1997 году консорциум W3C определил спецификацию RDF (Resource Description Framework). RDF предоставляет простой, но мощный язык описания ресурсов, основанный на триплетах (triple-based) «Субъект-Предикат-Объект» и спецификации URI. В 1999 году RDF получает статус рекомендации. Этот шаг в направлении улучшения функциональности и обеспечения интероперабельности (т.е. возможности обмениваться данными несмотря на их разнородность) в Сети считается одним из важнейших. Концептуально RDF дает минимальный уровень для представления знаний в Сети. Спецификация RDF опирается на ранние стандарты, лежащие в основе Web:

- Unicode, который служит для представления символов алфавитов различных языков;
- URI (Uniform Resource Identifier) – унифицированный (единообразный) идентификатор ресурса, т.е. последовательность символов, идентифицирующая абстрактный или физический ресурс, который используется для определения уникальных идентификаторов ресурсов;
- XML и XML Schema, используемый для структурирования и обмена информацией, а также для хранения RDF (XML синтаксис RDF).

Кроме RDF был разработан язык описания структурированных словарей для RDF – RDF Schema (RDFS). Он предоставляет мини-

мальный набор средств для спецификации онтологий. RDFS получил статус рекомендации W3C в 2004 году.

В 2004 году статус рекомендации получил язык OWL (Web Ontology Language). Он имеет 3 диалекта (3 множества структурных единиц), используемых в зависимости от требуемой выразительной мощности. OWL фактически является надстройкой над RDF/RDFS и поддерживает эффективное представление онтологий в терминах классов и свойств, обеспечение простых логических проверок целостности онтологии и связывание онтологий друг с другом (импорт внешних определений). Многие формализмы описания знаний могут быть отображены на формализм OWL (один из его диалектов – OWL DL – основан на дескриптивной логике). Большое число создаваемых в настоящее время онтологий кодируются на OWL, уже существующие онтологии транслируются в него.

В 2005 году началась работа над форматом обмена правилами – RIF (Rule Interchange Format). Его назначение – соединить в одном стандарте несколько формализмов для описания правил (по которым может осуществляться нетривиальный логический вывод): логики высших порядков, продукционные модели и т. п.

Уровням «Ontology vocabulary» и «Logic» соответствуют OWL и RIF. Для уровней «Proof» и «Trust» на данный момент не существует никаких стандартов. Здесь и возникает одно из существенных препятствий к реализации всей идеи: поддержка автоматической проверки корректности и правдивости информации.

3.3. Архитектура метаданных в World Wide Web. Документы, метаданные, связи

Ссылка URL открывает точное местоположение веб-ресурса в интернете. Часто под ресурсом понимается документ, поскольку в Сети много читабельных (удобных для чтения человеком) докумен-

тов – HTML-страниц, PDF-документов и т.п. Иногда ресурс – это просто некий объект, когда полученный ресурс имеет машинопонятный вид или обладает скрытым внутренним состоянием.

В рамках этого раздела термины «ресурс», «объект» и «документ» являются синонимами.

Неотъемлемой характеристикой любого ресурса Сети является сопровождающая его информация. Эту «сверхинформацию», или информацию об информации (о ресурсе), принято называть метаданными.

Под метаданными будем понимать машинопонятную информацию о веб-ресурсах и других сущностях.

Термин «машинопонятная» является ключевым. Речь идет о понимании информации программными агентами. Причем «понимании» с одной целью – использовать информацию для решения задач, возложенных на них (агентов) пользователем.

Метаданные должны иметь хорошо определенную ясную структуру и семантику [9].

Пример 1. Метаданные

Объект, извлеченный из сети по протоколу HTTP, может иметь дополнительную информацию (метаданные):

- дата создания или дата прекращения действия;
- владелец;
- другая информация.

Таким образом, в Сети есть данные – ресурсы, есть метаданные – информация о ресурсах. Эта информация, в свою очередь, тоже может рассматриваться как данные (ресурс).

Приведем две аксиомы (A1 и A2), на которых основана архитектура метаданных Сети.

Аксиома 1 (A1). Метаданные – это данные (или информация об информации – это тоже информация)

Поскольку метаданные – это данные, то они могут храниться в ресурсе (могут быть представлены как ресурс). То есть любой ресурс Сети может хранить как данные, так и метаданные о себе или о других ресурсах. На практике в Сети существует 3 способа передачи/получения метаданных:

1. Метаданные хранятся и передаются внутри документа (тег HEAD в HTML, данные о документе MS Word).
2. Сообщение метаданных происходит в процессе HTTP передачи (GET, POST или PUT).
3. Метаданные хранятся в каком-то другом документе.

Итак, метаданные могут храниться внутри документа, внутри другого документа либо передаваться вместе с документом средствами протокола HTTP.

Форма метаданных

Метаданные состоят из высказываний о данных и при представлении имеют форму имени (или типа высказывания) и набора параметров.

Аксиома 2 (A2). Архитектура, представляемая метаданными, является набором независимых высказываний (утверждений).

Как следствие, при группировке двух и более высказываний об одном ресурсе они объединятся логическим «И». Альтернативные высказывания являются независимыми, а их наборы представляют собой неупорядоченные множества. Конечно, высказывания можно комбинировать и другим способом, используя сложные синтаксические правила, но основной формой представления является неупорядоченный список, элементы которого связаны логическим «И».

Наиболее распространенной формой высказывания является следующая модель: Ресурс – атрибут – значение.

Здесь ресурс – это объект, о котором фиксируется высказывание, атрибут – некоторое свойство или параметр объекта, значение

представляет некоторое значение из области значений атрибута (или диапазона значений атрибута данного объекта).

Пример 2. Модель «Ресурс – атрибут – значение»

E-mail – Date – 01.01.2006; E-mail – From – Vasya;

В общем виде высказывание может быть выражено так:

(A u1 p q …),

где A – имя (или идентификатор) типа высказывания (Author, Date, …), u1 – URI ресурса, о котором делается высказывание, p, q, … – другие параметры, зависящие от типа высказывания, в том числе и представляющие значение атрибута.

Пространство имен атрибутов

Значения атрибутов и отношений могут сильно варьироваться, они могут задаваться спецификацией архитектуры или протокола. Но значения атрибутов могут быть определены для нужд одного конкретного приложения. Поэтому набор отношений и имен атрибутов должен быть легко расширяемым, следовательно, он должен быть расширяемым децентрализовано. Пространство URL подходит для определения имен атрибутов.

Пример 3. Словари с именами атрибутов

- HTML-элементы внутри элемента HEAD;
- заголовки HTTP-запроса, уточняющие атрибуты объекта.

Оба словаря определены внутри конкретных спецификаций, написанных на английском языке.

Связи

Отношение между двумя ресурсами будем называть связью.

Связь представляется тройкой (A u1 u2),

где A – тип отношения, u1 – URI первого ресурса, u2 – URI второго ресурса.

Связи являются основой навигации в Сети. Они могут использоваться для построения структур внутри www, а также для создания семантической Сети, в которой могут быть представлены знания об окружающем мире.

Иными словами, связи могут применяться для определения структуры данных (в этом случае они являются метаданными), но могут быть использованы и как форма представления данных.

Связи, как и прочие метаданные, могут быть переданы тремя (указанными выше) способами.

Одна из основных задач, решаемых при проектировании архитектуры метаданных Сети, состоит в том, чтобы сделать информацию самоописывающейся (self-describing).

Однако узким местом системы всегда является способ определения семантики метаданных и данных, применяемых внутри системы. Например, семантика метаданных заголовков e-mail и HTTP-сообщений определяется вручную на английском языке в виде спецификаций соответствующих протоколов. Эта семантика понятна людям (конечно, тем, кто знает английский). Чтобы теперь перейти к семантике, понятной машине, нужно использовать подходящий логический язык или язык представления знаний. Тогда семантика (точное значение) некоторого высказывания может быть выражена в терминах других отношений (более абстрактных концептов логического языка).

Преимущество самоописывающейся информации состоит в том, что нет необходимости согласовывать значение каждого термина централизованно, стандартизировать семантику высказываний. Язык RDF позволяет описывать метаданные о любых ресурсах Сети (и даже о сущностях, находящихся за ее пределами).

3.4. RDF

RDF – язык представления информации о ресурсах WWW. В частности, RDF служит для представления метаданных, связанных с ресурсами Сети, таких как «заголовок», «автор», «дата последнего изменения страницы». При увеличении объема метаданных, усложнении их структуры управление метаданными, построенными на основе XML Schema, становится трудоемкой задачей, для решения которой и предназначен RDF.

Модель данных RDF. RDF-граф

Базовой структурной единицей RDF является коллекция троек (или триплетов), каждая из которых состоит из субъекта, предиката и объекта (S, P, O). Набор триплетов называется RDF-графом (Рис). В качестве вершин графа выступают субъекты и объекты, в качестве дуг – предикаты (или свойства). Направление дуги, соответствующей предикату в данной тройке (S, P, O), всегда выбирается так, чтобы дуга вела от субъекта к объекту.

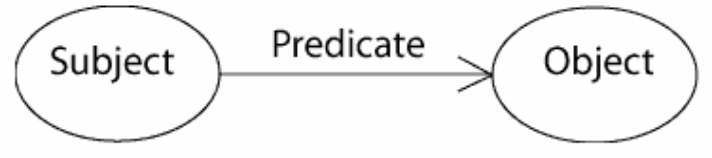

Рис. 9. RDF-тройка

Каждая тройка представляет некоторое высказывание, увязывающее S, P и O. Первые два элемента RDF-тройки (Subject, Predicate) идентифицируются при помощи URI.

Объектом может быть как ресурс, имеющий URI, так и RDF-литерал (значение).

RDF-литералы (или символьные константы)

RDF-литералы бывают двух видов: типизированные и не типизированные.

Каждый литерал в RDF-графе содержит 1 или 2 именованные компоненты.

Все литералы имеют лексическую (словарную) форму в виде строки символов Unicode.

Простые литералы имеют лексическую форму и необязательную ссылку на язык (ru, en…).

Типизированные литералы имеют лексическую форму и URI типа данных в форме RDF URI. Значение типизированного RDF-литерала определяется применением отображения. Это отображение определяется по URI типа данных и зависит от самого типа.

4. ЯЗЫКИ ПРЕДСТАВЛЕНИЯ ОНТОЛОГИЙ: RDFS, OWL. ЯЗЫК ЗАПРОСОВ SPARQL

Языки, о которых пойдет речь в данном разделе, являются основными языками Семантической Сети (Semantic Web). На сегодняшний день наблюдается разрыв между способами представления метаданных (языками их определения) и теми интеллектуальными агентами, которые должны ими пользоваться. Языки описания метаданных и онтологий в Web развиты очень хорошо, языки запросов и языки описания правил доведены до стадии технологических стандартов в данной области. Однако узким местом всё еще являются механизмы взаимодействия агентов на основе онтологий.

Многие популярные редакторы онтологий используют в качестве основного формализма дескриптивную логику (DL) и предоставляют средства для создания OWL-онтологий.

4.1. RDF (RDFS)

RDF модель данных, составляющая основу методики Semantic Web, является представителем семейства ER-моделей данных, специфика которой состоит том, что ресурсы и свойства идентифицируются с помощью глобальных идентификаторов – URI. RDF описывает предметную область в терминах ресурсов, свойств ресурсов и значений свойств. RDF-данные можно расценивать как совокупность утверждений – субъект, предикат (свойство) и объект утвер-

ждения, представлять в виде направленного графа, образуемого такими утверждениями.

RDF/XML-синтаксис [13, 14] позволяет записать граф в последовательной форме, пригодной для обмена данными. Этот синтаксис достаточно гибок – он допускает различные формы записи одного и того же графа, различные сокращенные формы.

RDF-схема (RDFS) [13, 14] представляет собой систему типов для Semantic Web. RDFS позволяет определить классы ресурсов и свойства как элементы словаря, и специфицировать, какие свойства, с какими классами могут быть использованы. RDFS выражает эти словари средствами RDF, предоставляя набор предопределённых ресурсов и свойств с обозначенной для них смысловой нагрузкой, которые могут быть использованы для описания новых RDF-словарей.

Таким образом, любое RDFS-описание представляет собой «обычные» RDF-данные – данные о классах и свойствах. RDFS позволяет определить уникальные (идентифицируемые URI) классы ресурсов, представляющие концептуальную модель конкретной предметной области, и уникальные (идентифицируемые URI) свойства, интересующие нас в этой области. Принадлежность ресурса к конкретному классу задается с помощью свойства rdf:type. Описываемые в словаре классы сами являются экземплярами предопределённого класса rdfs:Class, свойства же являются экземплярами rdf:Property. RDFS позволяет указать, каким классам присущи заданные свойства, и ресурсы какого класса могут появиться в качестве значения заданного свойства. Эта информация указывается в словаре с помощью свойств rdfs:domain и rdfs:range соответственно. RDFS позволяет связать классы (rdfs:Class) отношениями множественного наследования (rdfs:subClassOf). В RDFS-модели, как и в обычном объектном подходе, классам свойственен полиморфизм. То есть, экземпляр подкласса всегда может сыграть роль экзем-

пляра своего суперкласса, и появиться как субъект или объект свойства, для которого в качестве соответственно range или domain был указан суперкласс. Свойства также могут быть связаны отношениями множественного наследования (rdfs:subPropertyOf). Наследование свойства означает более узкую специализацию этого свойства, уточнение смысла и сужение границ использования.

Язык RDFS предоставляет лишь базовые возможности для описания словарей предметных областей, но он легко может быть расширен дополнительными примитивами моделирования, более детально и специализировано описывающими нужные аспекты классов и свойств. Механизм расширения внутренне присущ RDFS, поскольку для описания схем используется модель данных RDF, которая позволяет расширить описание любых ресурсов дополнительной информацией. Предопределённый словарь «мета-типов» RDFS также может быть расширен под нужды приложения, благодаря чему появляется возможность добавлять в язык новые примитивы.

Расширяемость позволяет RDFS стать фундаментом для более богатых языков концептуального моделирования – языков описания web-онтологий предметных областей. Цель таких языков – указать дополнительную машинно-интерпретируемую с-мантику ресурсов, то есть сделать машинное представление данных более похожим на положение вещей в реальном мире. Использование богатых языков концептуального моделирования позволит адаптировать к Web большое количество наработок в области систем инженерии знаний и баз знаний. Привлечение к Web систем логики и искусственного интеллекта составляет вершину «пирамиды Semantic Web», обеспечивая адекватный поиск информации и её машинную интерпретацию.

Первыми предложениями по описанию онтологий на базе RDFS были DAML-ONT (DARPA Agent Markup Language) [13] и European Commission OIL (Ontology Inference Layer) [14]. На базе

этих двух предложений возникло совместное решение – DAML+OIL, которое привело к созданию в рамках инициативы Semantic Web отдельной группы, ответственной за пересмотр этого решения и стандартизацию языка описания Web-онтологий (OWL – Web Ontology Language).

Однако ориентированность языков описания онтологий на системы математической логики делает их слишком тяжеловесными для огромного количества приложений, которым достаточно простого языка описания словарей – RDFS. И это правильно, каждая ступень в пирамиде Semantic Web – это ступень, на которой многие приложения могут остановиться, согласно своим собственным требованиям к данным и их использованию.

Модель данных RDF является синтаксически нейтральным способом представления RDF выражений [15, 16]. Модель данных RDF определяет простую модель для описания взаимосвязей между ресурсами с точки зрения названных свойств и значений. RDF-свойства можно рассматривать как атрибуты ресурсов и в этом смысле они соответствуют традиционным парам атрибут-значение. RDF-свойства также представляют собой отношения между ресурсами. Поэтому модель данных RDF напоминает диаграмму сущность-связь. Модель данных RDF, однако, не предусматривает никаких механизмов для определения этих свойств, а также никаких механизмов для определения отношений между этими свойствами и другими ресурсами. Это роль RDF-схемы.

С помощью описания ресурсов можно сообщить определенную информацию об определенных видах ресурсов. Для описания библиографических ресурсов, задаются универсальные описательные атрибуты, например, «автор», «название» и т.д. Для цифровой сертификации необходимы такие атрибуты как «контрольная сумма» и «авторизация. Объявление этих свойств (атрибутов) и соответствующей им семантики определены в контексте RDF как RDF-схема.

Схема определяет не только свойства ресурса (например, название, автор, тема, размер, цвет и т.д.), но может также определить виды описанных ресурсов (книги, веб-страницы, люди, компании и т.д.).

Спецификация RDFS не определяет словарь описательных элементов, таких как «автор», а определяет механизмы, необходимые для определения таких элементов, чтобы определить классы ресурсов, которыми они могут быть использованы, чтобы ограничить возможные комбинации классов и отношений, а также для выявления нарушений этих ограничений. Таким образом, механизм RDF Schema предоставляет базовую систему типов для использования в моделях RDF. Он определяет ресурсы и свойства, такие как [RDFS: Class] и [RDFS: subClassOf], которые используются при определении схем конкретных приложений.

RDF был разработан для описания метаданных для ресурсов Интернета, например, «утверждений», «ресурсов» и «свойств». Утверждения в RDF описывают ресурсы, которые могут быть веб-страницами или объектами реального мира, такими как публикации, люди или учреждения. Ресурсы и свойства описываются с использованием RDFS (RDF Schema). RDF Schema расширяет RDF за счет добавления новых примитивов моделирования, часто встречающихся в языках онтологий, таких как класс, наследование классов, наследование свойств, домен, ограничение диапазона.

RDF Schema обепечивает представление классов, слотов и фацетов, а RDF допускает представление экземпляров и фактов. В таком языке невозможно представить аксиомы, поэтому механизм логического вывода основан на отношениях между классами.

Преимуществом RDF является использование пространства имен XML и URI для идентификации сущностей, уже определенных в WWW. Это обеспечивает следующие возможности:

- В выражениях можно ссылаться на различные онтологии, распределенные в WWW.

- Онтологию можно дополнить в соответствии с существующими онтологиями, доступными в WWW.

Это позволяет обмениваться знаниями посредством Web и повторно использовать знания для определения новых онтологий.

4.1.1. Классы и свойства

В этом разделе описывается ядро схемы RDF: сначала определяется система типов, а затем вводятся основные классы и свойства.

RDF Schema, определенная в данном описании, представляет собой набор RDF ресурсов, которые могут быть использованы для описания свойств других ресурсов RDF (включая свойства), определяющих специфические для приложений RDF словари. Ядро схемы словаря, определенное в пространстве имен, неформально называемом «RDFS», получает официальный URI в окончательной версии данной спецификации.

Как описано в модели RDF и спецификации синтаксиса, ресурсы могут быть экземплярами одного или нескольких классов, это указывается с помощью rdf:type property. Классы часто организуются в иерархическую структуру, например, класс «собака» может рассматриваться как подкласс класса «животные», который является подклассом класса «организм» и т.д. Это означает, что любой ресурс, который имеет rdf:type «собака» фактически имеет также rdf:type «животное» и т.д. Данная спецификация описывает свойство rdfs:subClassOf для обозначения таких отношений между классами.

В дополнение к свойствам rdfs:subClassOf property, спецификация вводит ряд других ресурсов для определения ограничений на последовательное использование свойств и классов в данных RDF. Например, схема RDF может описать ограничения на типы значений некоторых свойств или на классы, для которых следует определить такие свойства. Спецификация дает механизм для описания та-

ких ограничений, но не определяет, как приложение должно обрабатывать информацию об ограничении. Например, RDF схема определяет, что «книга» может иметь свойство «автор», но не определяет, как приложение должно действовать при обработке этой информации. Различные приложения будут использовать ограничения различными способами, например, валидатор будет искать ошибки, интерактивный редактор – предлагать корректные значения.

Наборы классов соответствуют наборам типов данных. Спецификация не определяет типы данных, но тип данных может быть использован в качестве значения rdfs:range property.

Рис. 10 иллюстрирует концепции класса, подкласса и ресурса. Класс изображается прямоугольником с закругленными углами, ресурс изображается большой точкой. Стрелки ведут от ресурса к классу, который он определяет. Подкласс показан прямоугольником с закругленными углами, который полностью включен в другой класс (супер-класс). Если ресурс находится внутри класса, существует явное или неявное свойство rdf:type property этого ресурса, значением которого является ресурс, определяющий внутренний класс. (Эти свойства показаны в виде дуг в ориентированном размеченном графе, представленном на рис. 11). Ресурсы RDF, представленные на рис. 11, описаны в модели RDF и спецификации синтаксиса.

Рис. 11 показывает ту же информацию об иерархии классов, что и рис. 10, но в виде графического представления модели данных RDF с использованием «узлов» и «дуг». Если класс является подмножеством другого класса, то существует дуга rdfs:subClassOf, направленная из узла, представляющего первый класс, к узлу, представляющему второй класс. Аналогично, если ресурс является экземпляром класса, то существует дуга rdf:type, направленная из ресурса в узел, представляющий класс.

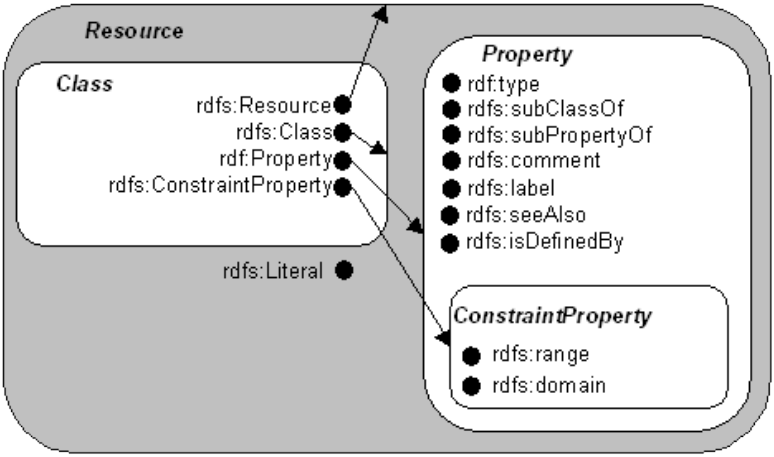

Рис. 10. RDF-тройка

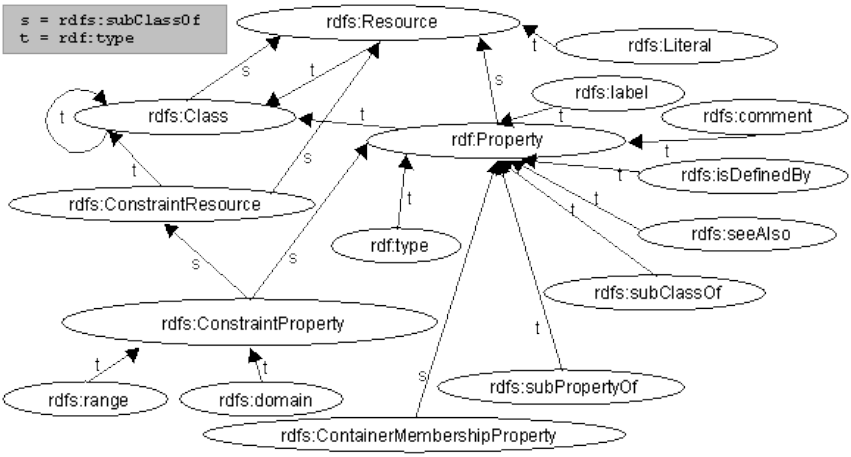

Рис. 11. Иерархия классов в RDF Schema

Ниже приведен пример использования RDF и RDF Schema.

Заголовок RDF файла:
<rdf:RDF xml:lang=”en”
xmlns : rdf = http://www.w3.org/1999/02/22-rdf-syntax-ns#
xmlns: rdfs = http://www.w3.org/2000/01/rdf-schema#>

Описание концептов ServiceProvider и Operation как подклассов встроенного класса RESOURCE:
<rdf:Description ID=”ServiceProvider”>
<rdf:type resource=htMtipr:i//www.w3.aomrg/2000/01/rdf-schema#Class/>
<rdfs:subClassOf rdf:resource=http://www.w3.org/2000/01/rdf-schema#Resource/>
</rdf:Description>
<rdf:Description ID=”Operation”>
<rdf:type resource=http://www.w3.org/2000/01/rdf-schema#Class/>
<rdfs:subClassOf rdf:resource=http://www.w3.org/2000/01/rdf-schema#Resource/>
</rdf:Description>

Описание концептов Business_Operation, Transportation_Operation и Manufacturing_Operation как подклассов класса Operation:
<rdf:Description ID=”Business_Operation”>
<rdf:type resource=http://www.w3.org/2000/01/rdf-schema#Class/>
<rdfs:subClassOf rdf:resource=”#Operation”/>
</rdf:Description>
<rdf:Description ID=”Transportation_Operation”>
<rdf:type resource=http://www.w3.org/2000/01/rdf-schema#Class/>

<rdfs:subClassOf rdf:resource=”#Operation”/>
</rdf:Description>
<rdf:Description ID=”Manufacturing_Operation”>
<rdf:type resource=http://www.w3.org/2000/01/rdf-schema#Class/>
<rdfs:subClassOf rdf:resource=”#Operation”/>
</rdf:Description>
Описание свойств класса ServiceProvider:
<rdf:Description ID=”provide_service”>
<rdf:type resource=http://www.w3.org/1999/02/22-rdf-syntax-ns#property/>
<rdfs:domain rdf:resource=”#Service_Provider”/>
<rdfs:range rdf:resource=”http://www.myhome.com/Ontology_Operation#Operation”/>
</rdf:Description>
<rdf:Description ID=”service_provider_Identification”>
<rdf:type resource=http://www.w3.org/1999/02/22-rdf-syntax-ns#property/>
<rdfs:domain rdf:resource=”#Service_Provider”/>
<rdfs:range rdfs:Literal”/>
</rdf:Description>
<rdf:Description ID=”has_capability”>
<rdf:type resource=http://www.w3.org/1999/02/22-rdf-syntax-ns#property/>
<rdfs:domain rdf:resource=”#Service_Provider”/>
<rdfs:range rdf: resource=”http://www.myhome.com/Ontology_Process#Process”/>
</rdf:Description>
<rdf:Description ID=”has_manufacturing_capability”>
<rdf:type resource=http://www.w3.org/1999/02/22-rdf-syntax-ns#property/>

<rdfs:subPropertyOf rdf:resource=”#has_capability”/>
</rdf:Description>

Пример утверждения в RDF:

```
<rdf:RDF xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#>
<rdf:Description ID=”TaxiService”>
<rdf:type rdf:ID=”#Service_Provider”/>
<provide_service rdf:resource=”#People_Transportation”/>
<Service_Provider_Identification>kl-124587-
om</Service_Provider_Identification >
<has_capability rdf:resource=”#Secured-Web-Payment”/>
<Contact_information>http://wwww.bluetaxi.com</Contact_infor-
mation>
</rdf:description>
<rdf:Description ID=”People_Tranportation”>
<rdf:type rdf:ID=”#Tranportation_Operation”/>
</rdf:description>
<rdf:Description ID=” Secured-Web-Payment”>
<rdf:type rdf:ID=”#Business_Process”/>
</rdf:description>
</rdf:RDF>
```

4.1.2. Возможности и ограничения языка RDF (RDF Schema). Сопоставление RDF(S) с другими парадигмами

Система типов RDFS похожа на многие общепринятые системы типов, как в ER-моделировании, объектно-ориентированном программировании и UML и т.п. Инициатива Semantic Web не ставит перед собой цели создать новую модель данных, напротив, она ориентируется на интеграцию различных моделей данных с целью получения информации из соответствующих источников.

RDFS отличается от этих стандартных систем типов в нескольких существенных аспектах, которые являются следствием глобализации и децентрализации информационной системы, к которой мы приходим, «выходя» в Web из установленных моделью данных рамок. В каком-то смысле RDF(S) есть адаптация этих моделей к Web. Рассмотрим сопоставление примитивов RDFS и модели данных объектно-ориентированного программирования [17].

Один из архитектурных принципов Web состоит в том, что кто угодно может расширить описание существующих ресурсов, то есть «кто угодно может сказать, что угодно, о чём угодно». Это означает, что отношение между двумя объектами может храниться отдельно от любой другой информации об этих объектах. Это сильно отличается от того, к чему мы привыкли в обычных объектно-ориентированных системах, в которых считается, что информация об объекте хранится внутри объекта: определение класса объекта подразумевает указание места хранения его свойств. Такое отличие является следствием децентрализации и адаптации к положению вещей в реальном мире. Например, один человек может определить автомобиль, как нечто, имеющее колёса, вес и размер, но не предвидеть цвет. Это не остановит другого человека от утверждения, что его машина – красная, используя некоторый словарь цветов.

Из этого архитектурного принципа Web следует основное отличие парадигмы RDFS от объектной парадигмы – это ее свойство – центричность. Свойства (отношения, предикаты) в RDFS являются объектами первого уровня, как и классы: они идентифицируются URI и определяются независимо от классов, тогда как в объектной и ER парадигмах свойства (атрибуты) указываются в «теле» класса, смысл свойств с одинаковыми названиями в разных классах может быть различен.

Вместо того чтобы описывать классы в терминах свойств (структуры), имеющихся у него, как это делается в объектно-ори-

ентированных системах, RDFS описывает свойства в терминах классов, к которым они применимы, указывая rdfs:domain (область применения свойства) и rdfs:range (область значений свойства). Различие между этими подходами может показаться только синтаксическим, но на самом деле есть существенная разница, которая связана как раз с глобализацией информационной системы при адаптации её к Web, где «кто угодно может сказать, что угодно, о чём угодно». Например, если кем-то определен класс ex:Book со свойством ex:author, принимающим значения типа ex:Person, то это не запрещает другим разработчикам придать классу ex:Book дополнительное свойство my:publisher, достаточно лишь указать этот класс в rdfs:domain нового свойства my:publisher. Это не требует переопределения класса, причем создатели класса могут быть в неведении данного факта. В то же время в ООП потребовалось бы пе-реопределить и перекомпилировать класс.

Кроме того, RDFS вообще не требует, чтобы у свойства была задана область применения – свойство без domain может быть использовано для описания любого ресурса, независимо от его класса. Определение свойства без указания области применения позволяет использовать его в будущем в ситуациях, которые не могли быть предвидены в момент разработки схемы. Именно так поступает Dublin Core [18], предоставляя словарь стандартных свойств, пригодных для описания любого Web-ресурса, для которого они окажутся полезными.

Другое важное отличие в семантике RDFS-описаний – это то, что они носят описательный, а не «предписывающий» характер, то есть, они могут использоваться не для того, чтобы наложить ограничения на применение свойств, а просто чтобы предоставить дополнительную информацию приложению, обрабатывающему эти данные. Если ОО язык программирования объявляет класс Book с атрибутом author типа Person, это обычно интерпретируется как

набор ограничений (условий применения). ОО язык не позволит создать экземпляр класса Book без атрибута author или указать в качестве значения author объект, не являющийся экземпляром Person. Наконец, он не позволит создать экземпляр Book с каким-то другим атрибутом.

RDF Schema, напротив, предоставляет информацию о схеме как дополнительное описание ресурсов, но не указывает, как это описание должно использоваться приложениями. Приложение вольно по своему усмотрению считать RDF-данные соответствующими схеме или нет, если в описании отсутствует некоторое свойство, требуемое схемой, либо присутствуют свойства, не указанные в схеме. Одно приложение может интерпретировать RDFS-описания как шаблон для генерации данных, и про-верять соответствие данных областям значений свойств, то есть интерпретировать описания схемы так же, как они интерпретируются в ОО языке программирования. Другое приложение может интерпретировать RDFS-описания как дополнительную информацию о данных, которые оно получает. Например, если оно получит RDF-данные с указанием свойства ex:author, содержащим значение без указания типа, то может заключить на основе описания схемы, что это значение является ex:Person. Третье приложение может получить данные, в которых свойство ex:author содержит ресурс типа ex:Student, и использовать информацию схемы как базис для предупреждения, что данные могут содержать ошибку. Хотя, возможно, где-то существует RDFS-описание, решающее эту проблему, например, указывающее, что ex:Student подкласс ex:Person.

Итак, RDFS утверждения всегда описательны. Они могут, конечно, интерпретироваться как «предписывающие» (ограничения), но только если приложение желает их так интерпретировать. Всё, что делает RDFS-описание – это предоставляет приложениям дополнительную информацию «для размышления».

Преимущества RDF

К преимуществам RDF следует отнести:

- обобщенный способ работы с метаданными;
- ориентация на программное обеспечение в качестве конечного потребителя информации;
- возможность осуществлять автоматическую обработку Web-ресурсов:
- поиск;
- каталогизация;
- генерация иерархических карт сайтов.

Недостатки RDF

Открытость и расширяемость RDF ведет к тому, что «кто угодно» (т. е. любой пользователь RDF) может сказать «что угодно» (т. е. фиксировать произвольное утверждение) о «чем угодно» (т. е. о любом ресурсе), используя RDF. Следовательно, нет никакой гарантии целостности и непротиворечивости RDF-описаний. Вся ответственность за проверку ложится на получателей (конечных пользователей) метаданных, т. е. на разработчиков приложений, обрабатывающих RDF-данные.

Из вышесказанного о RDF и метаданных можно сделать вывод, что RDF имеет довольно слабые (по объему) выразительные средства и не основан на каком-либо логическом формализме. Это язык описания метаданных, причем метаданных в широком смысле слова: имеющих произвольную структуру и смысл.

RDF – универсальный инструмент и поэтому требует настройки для решения конкретных специализированных задач. Способ такой «настройки» состоит в расширении RDF при помощи словарей. Рассмотрим одно из расширений RDF для области проектирования и представления онтологий.

4.2. OWL

Язык W3C Web Ontology Language (OWL) является языком Semantic Web, предназначенным для представления сложных знаний о понятиях, группах понятий и отношениях между понятиями. OWL основан на вычислительной логике, поэтому знания, выраженные с помощью OWL, могут быть использованы компьютерными программами, например, для проверки согласованности этих знаний или чтобы сделать неявное знание явным. OWL документы, известные как онтологии, могут быть опубликованы в World Wide Web, могут ссылаться на другие OWL онтологии или могут быть использованы в других OWL онтологиях [19, 20].

OWL (Web Ontology Language) – язык представления онтологий в Web. Фактически это словарь, расширяющий набор терминов, определенных RDFS. OWL-онтологии могут содержать описания классов, свойств и их экземпляров. Создание OWL – это ответ на необходимость представления знаний в Сети в едином формате. Исторически предшественником OWL был язык DAML+OIL, объединивший 2 инициативы: проект DAML (DARPA Agent Markup Language) и проект OIL (Ontology Inference Layer). Наиболее ранним проектом представления онтологий в Web был SHOE (Simlpe HTML Ontology Extensions). Ветви развития языков описания онтологий для Web показаны на рис. 12. Верхний уровень: OIL, DAML+OIL и OWL продолжают развиваться, но наибольшей популярностью пользуется OWL.

OWL с 2004 года является рекомендацией W3C и объединяет лучшие черты своих предшественников.

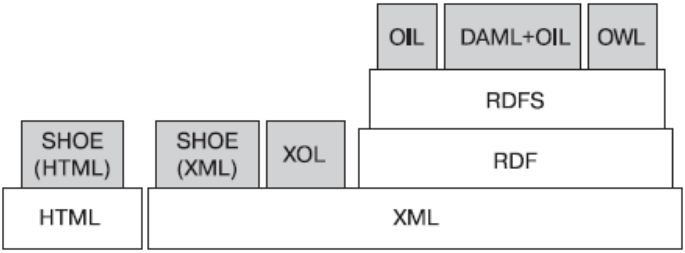

Рис. 12. Основные ветви развития языков описания онтологий для Web

4.2.1. Структура OWL-онтологии. Базовые элементы OWL

Любая онтология имеет заголовок и тело. В заголовке содержится информация о самой онтологии (версия, примечания), импортируемых онтологиях. За заголовком следует тело онтологии, содержащее описания классов, свойств и экземпляров.

Классы

В OWL введен новый термин – «класс» (owl:Class). Необходимость этого объясняется тем, что не все классы диалектов DL и Lite являются rdfs-классами (в этом случае owl:Class является подклассом rdfs:Class). В диалекте Full подобных ограничений нет, и owl:Class фактически является синонимом rdfs:Class.

Для организации классов в иерархию используется свойство rdfs:subClassOf.

Особое место занимают два взаимодополняющих класса – Thing и Nothing. Первый из них является надклассом любого класса OWL, второй – подклассом любого класса OWL. Экземпляр любого класса OWL входит в экстенсионал класса Thing. Экстенсионал класса Nothing является пустым множеством.

OWL-класс может быть описан шестью способами:

1) идентификатором класса (URI);
2) перечислением всех экземпляров класса;
3) ограничением на значение свойства;
4) пересечением 2-х и более определений классов;
5) объединением 2-х и более определений классов;
6) дополнением (логическим отрицанием) определения класса.

Только первый способ определяет именованный класс OWL. Все оставшиеся определяют анонимный класс через ограничение его экстенсионала. Способ 2 явно перечисляет экземпляры класса, способ 3 ограничивает экстенсионал только теми экземплярами, которые удовлетворяют данному свойству. Способы 4-6 используют теоретико-множественные операции (объединение, пересечение и дополнение) над экстенсионалами соответствующих классов, чтобы определить экстенсионал нового класса.

Описания класса формируют строительные блоки для определения классов через аксиомы.

Простейшая аксиома, определяющая именованный класс (существование класса с именем «Human»):

<owl:Class rdf:ID="Human"/>.

В OWL определены еще 3 конструкции, комбинируя которые можно определять более сложные аксиомы классов:

- rdfs:subClassOf говорит о том, что экстенсионал одного класса (подкласса) полностью входит в экстенсионал другого (надкласса);
- owl:equivalentClass говорит о том, что экстенсионалы двух классов совпадают;
- owl:disjointWith говорит о том, что экстенсионалы двух классов не пересекаются. Иногда говорят, что таким образом определяются дизъюнктивные классы.

Свойства

В OWL выделяют две категории свойств: свойства-объекты (или объектные свойства) и свойства-значения. Первые связывают между собой индивиды (экземпляры классов). Вторые связывают индивиды со значениями данных. Оба класса свойств являются подклассами класса rdf:Property.

Для определения новых свойств как экземпляров owl:ObjectProperty или owl:DatatypeProperty используются аксиомы свойств.

Пример аксиомы:

<owl:ObjectProperty rdf:ID="hasParent"/>

Данная аксиома объявляет существование некоторого свойства «hasParent», связывающего экземпляры класса owl:Thing друг с другом.

Кроме того, OWL поддерживает следующие конструкции для построения аксиом свойств:

- Конструкции RDF Schema: rdfs:subPropertyOf (определяет подсвойство данного свойства), rdfs:domain (определяет домен свойства) и rdfs:range (определяет диапазон свойства).
- Отношения между свойствами: owl:equivalentProperty (определяет эквивалентное свойство) и owl:inverseOf (определяет обратное свойство).
- Ограничения глобальной кардинальности: owl:FunctionalProperty (определяет однозначное свойство — однозначное отображение домена свойства на диапазон) и owl:InverseFunctionalProperty (обратно функциональное свойство, т. е. определяет, что свойство, обратное данному свойству, является однозначным).
- Логические характеристики свойства: owl:SymmetricProperty (определяет свойство как симмет-

ричное) и owl:TransitiveProperty (определяет транзитивное свойство).

Индивиды (экземпляры классов или свойств)

Индивиды определяются при помощи аксиом индивидов (т. н. фактов). Рассмотрим два вида фактов:

1) факты членства индивидов в классах и о значениях свойств индивидов;
2) факты идентичности/различности индивидов.

Пример аксиом индивида первого вида:

```
<Балет rdf:ID="ЛебединоеОзеро">
  <имеетКомпозитора rdf:resource="#Чайковский"/>
</Балет>
```

Данная аксиома определяет сразу 2 факта: (1) существует некоторый индивид класса «Балет», имеющий имя «ЛебединоеОзеро»; (2) этот индивид связан свойством «имеетКомпозитора» с индивидом: «Чайковский» (определенным где-то в другом месте).

Первый факт говорит о членстве в классе, второй – о значении свойства индивида.

Аксиомы второго вида необходимы для суждения об идентичности индивидов. Дело в том, что в OWL не делается никаких предположений ни о различии, ни о совпадении двух индивидов, имеющих различные идентификаторы URI.

Подобные утверждения выражаются аксиомами идентичности с помощью следующих конструкций:

- owl:sameAs постулирует, что две ссылки URI ссылаются на один и тот же индивид;
- owl:differentFrom постулирует, что две ссылки URI ссылаются на разные индивиды;
- owl:AllDifferent предоставляет средство для определения списка попарно различных индивидов.

На рис. 13 проиллюстрированы основные элементы OWL-онтологии.

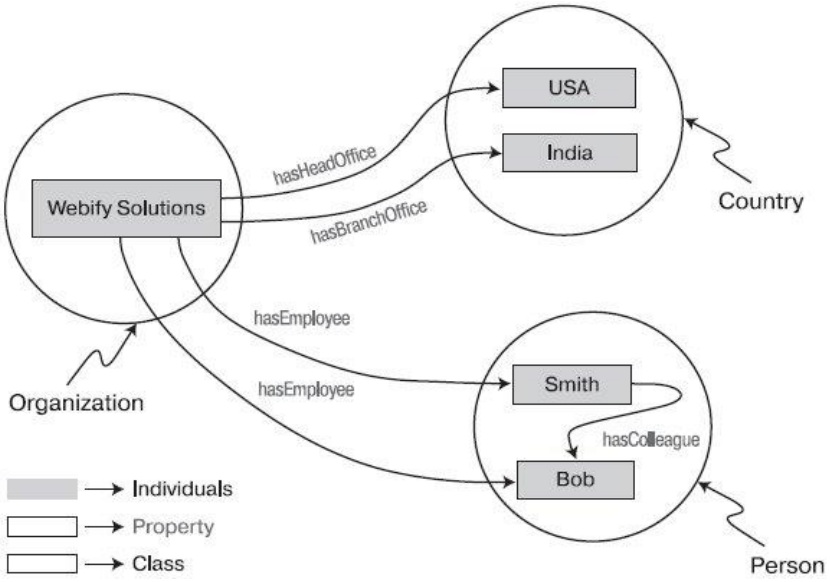

Рис. 13. Основные структурные единицы OWL-онтологии

4.3. SPARQL

SPARQL – это язык запросов к RDF-хранилищам. Синтаксически он очень похож на SQL [19, 20]. SPARQL (произносится «sparkle») – язык семантических запросов к базам данных, предназначенный для извлечения и обработки данных, хранимых в формате Resource Description Framework (RDF). Он был разработан как стандарт группой RDF Data Access Working Group (DAWG) консорциума World Wide Web Consortium и был признан в качестве одной из ключевых технологий Semantic Web. 15 января 2008 г. SPARQL 1.0 был объявлен официальной рекомендацией W3C.

SPARQL позволяет строить запросы, состоящие из RDF-троек, конъюнкции, дизъюнкции и необязательных шаблонов. Существуют реализации для нескольких языков программирования. Существуют инструменты, которые позволяют подключить и полуавтоматически построить запрос SPARQL для конечной точки SPARQL, например, ViziQuer. Кроме того, существуют инструменты, которые транслируют SPARQL запросы на другие языки запросов, например, SQL и XQuery.

SPARQL позволяет пользователям писать запросы к данным, которые соответствуют спецификации RDF W3C. Вся база данных при этом рассматривается как набор троек «субъект-предикат-объект». Это аналогично использованию в некоторых базах данных NoSQL, таких как MongoDB, терминов «документ-ключ-значение».

Данные RDF могут также рассматриваться в терминах реляционной базы данных SQL в виде таблицы с тремя колонками – субъект, предикат, объект. В отличие от реляционных баз данных, столбец «объект» является неоднородным: тип данных в ячейке, как правило, определяется (или указывается в онтологии) значением предиката. С другой стороны, по сравнению с отношениями SQL, вся тройка для данного субъекта может быть представлена в виде строки, где субъект является первичным ключом, каждый возможный предикат – столбцом, а объект – значением в ячейке. SPARQL/RDF становится более легким и более мощным для столбцов, которые могут содержать множественные значения (например, «дети») для одного и того же ключа, и где сам столбец может быть соединяемой переменной в запросе, а не непосредственно заданной.

SPARQL предоставляет полный набор аналитических операций для формирования запросов, таких как JOIN, SORT, AGGREGATE для данных, схема которых, по сути, является частью данных и не требует отдельного определения. Информация о схеме (онтология) часто оказывается внешней, что позволяет однозначно

объединять различные наборы данных. Кроме того, SPARQL предоставляет определенный синтаксис обхода графа для данных, которые можно представить в виде графа и рисунка.

4.3.1. Запросы SPARQL

В приведенном ниже примере показан простой запрос, который использует определение онтологии «foaf», часто называемой «friend-of-a-friend» онтологией.

В частности, следующий запрос возвращает имена и адреса каждого человека в наборе данных:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?email
WHERE {
  ?person a foaf:Person.
  ?person foaf:name ?name.
  ?person foaf:mbox ?email.
}
```

Этот запрос объединяет все тройки с соответствующим субъектом, где тип предиката – человек (foaf: Person), имеющий одно или несколько имен (foaf: имя) и почтовых ящиков (foaf: Mbox) ,

Автор этого запроса выбрал ссылку на субъект, используя имя переменной «?person» для ясности. Так как первый элемент тройки всегда субъект, можно было бы использовать любое имя переменной, например, «? Subj» или «?X». Какое бы имя ни было выбрано, оно должно быть одинаковым в каждой строке запроса, чтобы показать, что обработчик запроса должен присоединять тройки с тем же субъектом.

Результат объединения представляет собой набор строк – ?person, ?name, ?email. Этот запрос возвращает ?name и ?email, т.к. ?person часто является сложным URI, а не строкой, удобной для человека. Некоторые люди могут иметь несколько почтовых ящиков,

поэтому в возвращаемом наборе строка ?name может появляться несколько раз, один раз для каждого почтового ящика.

Для запросов, которые считывают данные из базы данных, язык SPARQL определяет четыре различных варианта запросов для разных целей.

SELECT query

Используется для извлечения исходных значений из конечной точки SPARQL, результаты возвращаются в виде таблицы.

CONSTRUCT query

Используется для извлечения информации из конечной точки SPARQL и преобразования результатов в корректный RDF.

ASK query

Используется для получения простого результата True / False для запроса в конечной точке SPARQL.

DESCRIBE query

Используется для извлечения графа RDF из конечной точки SPARQL, содержание которого остается в конечной точке, для принятия решения на основании того, что сопровождающий считает полезной информацией.

Каждая из этих форм запроса принимает WHERE блок для ограничения запроса, хотя в случае запроса DESCRIBE запрос WHERE не является обязательным.

SPARQL определяет язык для обновления базы данных с несколькими новыми формами запросов.

Пример

Запрос SPARQL, который моделирует вопрос «What are all the country capitals in Africa?»:

```
PREFIX ex: <http://example.com/exampleOntology#>
SELECT ?capital ?country
WHERE {
```

```
  ?x ex:cityname ?capital ;
    ex:isCapitalOf ?y .
  ?y ex:countryname ?country ;
    ex:isInContinent ex:Africa .
}
```

Переменные определяются с помощью префикса «?» или «$». «Связки» для ?capital и the ?country будут возвращаться.

Процессор запросов SPARQL будет искать наборы троек, которые соответствуют этим четырем «тройкам», связывающим переменные в запросе с соответствующими частями каждой тройки. Соответствие классов может определяться за счет атрибутов классов или свойств.

4.4. Контрольные вопросы по разделам №3–№4

1. Дайте определение семантического Web. Какова его цель?
2. Какие уровни включает многоуровневое представление (стек) семантического Web?
3. Что такое URI?
4. Что такое URL?
5. Что такое ресурс RDF?
6. Каково назначение метаданных? Что входит в состав метаданных?
7. Опишите форму представления метаданных.
8. Что определяет пространство имен атрибутов при описании метаданных?
9. Что определяют связи при описании метаданных?
10. Опишите модель данных RDF: цель, основная функциональность. Каким образом RDF описывает предметную область?
11. С какой целью используется RDF-схема (RDFS)? Основная функциональность.

12. Как определяются классы и свойства в RDFS?

13. Каким образом RDFS позволяет задать принадлежность ресурса к конкретному классу?

14. Что в RDFS определяют rdfs:domain и rdfs:range соответственно?

15. В чем заключаются достоинства и недостатки RDF и RDFS?

16. Перечислите требования, предъявляемые к языку онтологий для Semantic Web. Какие языки для описания онтологий были разработаны согласно этим требованиям?

17. Опишите структуру онтологии OWL.

18. Как описываются классы в OWL?

19. Как описываются свойства в OWL?

20. Как описываются индивиды в OWL?

21. Каково назначение языка SPARQL?

22. Какие виды запросов используются в SPARQL?

5. МОДЕЛЬ ОНТОЛОГИИ

Конечная цель создания и использования онтологий – обеспечить поддержку деятельности по накоплению, разделению и повторному использованию знаний предметной области или предприятия.

Исходя из этой цели, вводятся критерии, которым должна отвечать онтология:

- прозрачность – онтология должна эффективно передавать подразумеваемое значение определенного термина, необходимого для описания ситуаций;
- связность – онтология должна быть связной, т.е. она должна позволять делать выводы, которые согласуются с исходными определениями понятий. По крайней мере, определяемые аксиомы должны быть по возможности логически согласованы между собой, чтобы не вызывать множества противоречий;
- расширяемость – онтология должна быть разработана с возможностью использования разделяемого и пополняемого словаря, когда для формализации ситуации требуются новые и новые знания;
- независимость от синтаксиса – концептуализация должна быть специфицирована на уровне знания максимально независимо от представления понятий на уровне символов;
- удобство для пользователя – онтология должна позволять выражать знания в привычном для пользователя (лица, принимающего решения) виде, быть понятной, обозримой и связанной;

- эффективность машинной обработки – важна возможность формализации онтологий к виду, допускающему эффективную компьютерную обработку, чтобы снять с пользователя рутинные операции по поддержанию, использованию и развитию онтологий;
- минимальный базис при высокой выразительности – онтология должна вводить минимальный базовый набор понятий, но их должно быть достаточно, чтобы моделировать мир в требуемых целях и описывать сложные ситуации.

5.1. Онтологические базисы

Под формальной моделью онтологии O часто понимают упорядоченную тройку вида O = <C, R, F>, где C – конечное множество концептов (понятий) предметной области, которую определяет онтология O; R – конечное множество отношений между концептами (понятиями) предметной области; F – конечное множество функций интерпретации (аксиоматизация), заданных на концептах и/или отношениях онтологии O [21, 22]. Естественными ограничениями, накладываемыми на множество C, являются конечность и не пустота. Что касается множеств R и F, то они могут быть пустыми, что соответствует частным видам онтологии, когда она вырождается в простой словарь и таксономию понятий.

Таким образом, онтологии на базовом уровне должны, прежде всего, обеспечивать словарь понятий (терминов) для представления и обмена знаниями о предметной области и множество связей (отношений), установленных между понятиями в этом словаре.

Для формализации знаний в онтологии необходимо выбрать базис, в котором будут описываться концепты. В качестве примера одного из таких базисов в [23] предложен следующий набор компонент:

- классы (classes) – обычно организованы в таксономии;
- отношения (relations) – представляют тип связей между концептами предметной области. Формально они определяются как декартово произведение n множеств таких, что R: Cl x C2 x ... x Cn. Примеры простых бинарных отношений: «быть частью» («part-of»), «подкласс-класс» («subclass-of») или «связанный-с» («connected-to»);
- функции (functions) – специальный случай отношений, в котором n-й элемент отношения определяется по значениям (n-1) предшествующих элементов – F: Cl x C2 x ... x Cn-1 => Cn. Примеры: функция «Price-of-a-used-car», которая вычисляет цену подержанной машины в зависимости от ее модели, даты выпуска и числа километров;
- аксиомы (axioms) – моделируют предложения, которые всегда истинны. Пример аксиомы: «если что-то сделано из дерева, оно может гореть»;
- экземпляры (instances) – представляют элементы. Например, моя конкретная маленькая белая мышь является экземпляром класса «Мыши».

Из всего множества отношений в онтологии выделяется специальный класс – простая таксономия: O = T° = < C, {is_a}, {}>.

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

Заметим, что данный базис – не единственный, например, в [24] рассматривается вариант «объекты» – «отношения» – «роли» – «атрибуты» при построении онтологий организаций.

Еще одним примером онтологического базиса может служить формальная онтология свойств, представленная в [25], в которой свойство – центральная сущность. Здесь рассматривается, в первую очередь, проблема формирования правильной таксономической структуры онтологии. Свойство соответствует узлу таксономии. Формальная онтология основывается на наборе мета-свойств, построенных вокруг философских понятий идентичности (identity), единства (unity), сущности (entity) и зависимости (dependency). Комбинации мета-свойств определяют разновидности свойств такие, как категория, тип, роль, атрибут и прочие. Таким образом, мета-свойства налагают некоторые ограничения на категоризацию и помогают «очищать» таксономии, т.е. прояснять неправильно выделенные концепты (misconcepts), делая их более понятными, облегчая их сравнение и интеграцию. На практике, ответив на простые вопросы о каждом свойстве в иерархии («Присуще ли это свойство всем его экземплярам?», «Зависит ли от какого-нибудь другого свойства?» и т.п.), можно определить наборы мета-свойств каждого узла и, соответственно, отнести его к определенной разновидности свойств. Таким образом, строится основная линия наследования – backbone taxonomy, в которую входят категории, типы и квазитипы. Остальные свойства (различные атрибуты, роли и т.п.) становятся элементами боковых линий.

5.2. Онтология Аристотеля

В развитие идеи универсального базиса для представления знаний, где важной частью является созидательная деятельность людей предметной области, которая изначально рассчитана на представление процессов и действий пользователей, предложена концептуальная модель (метаонтология, или модель Аристотеля) [26].

Предлагаемая базовая модель позволяет описывать не только декларативные знания о предметной области, понятия и сущности

«мира», но и процедурные знания, деятельную компоненту, представляющую сценарии действий над объектами, выражающими законы мира, свойства или функции объектов, или действия субъекта над объектами мира. Кроме того, модель мира всегда предполагает наличие некоторой модели пространства и времени, в рамках которого существуют и взаимодействуют все объекты мира, над которыми можно выполнять действия [27].

Предлагаемая «Метаонтология Аристотеля» является общей для всех миров – как физических, так и абстрактных и включает следующие концепты:

- «объекты» – сущности, характеризуемые свойствами,
- «свойства», отражающие способность объектов вступать во взаимодействия,
- «процессы» – цепочки действий по изменению состояний объектов,
- «отношения», позволяющие связывать объекты и конструировать сложные объекты из простых,
- «атрибуты», характеризующие состояния концептов.
- Метаонтология Аристотеля предполагает следующие важные свойства:
- существуют объекты, которые обладают свойствами и характеризуются состояниями;
- с каждым объектом можно что-то делать, изменяя состояние, свойства или отношения между объектами;
- отношения между объектами могут отражать структурные, функциональные, временные или любые другие виды связей;
- чтобы выполнить действие над объектом, необходимо соблюдение определенных условий, которые задаются свойствами и отношениями;

- сложные объекты строятся из простых объектов путем выполнения действий (процессов) над ними для установления отношений и связывания простых объектов в сложные;
- свойства выражают способность объектов вступать в процессы взаимодействия на основе законов мира;
- события, действия (процессы) изменяют состояния объектов, их свойства и отношения и запускают новые процессы;
- процессы состоят из действий с объектами, так же как сложные объекты состоят из простых;
- с каждым объектом мира можно что-то делать в любой момент времени (нет тупиковых состояний), но чтобы выполнить действие над объектом, необходимо выполнение определенных условий;
- объекты, свойства, отношения, процессы (действия) характеризуются атрибутами различных типов, которые имеют диапазоны значений и конкретные значения в заданный момент времени;
- атрибуты объекта/отношения являются качественной или количественной характеристикой понятия;
- правила являются обобщенными понятиями для формализованных условий вида «если-то» (предикатов) и высказываний (утверждений, аксиом, фактов).

Данная модель предлагается как наиболее адекватная для описания сложных разнородных предметных областей [28, 29].

Ниже приведен перечень понятий и отношений на примере онтологии «РЖД»:

- классы понятий: «Транспортное средство», «Поезд», «Вагон», «Бригада», «Груз».
- примеры свойств: Цистерна «Может перевозить» нефть.
- примеры процессов: «Груз доставляется вагоном потребителю», «Вагоны формируют поезд».

- примеры отношений: Груз «находится» на поезде, груз «забронировал» вагон.
- примеры атрибутов поезда: «Плановая дата отбытия», «Длительность движения», «Станция назначения», «Скорость движения».

В упрощенном виде фрагмент онтологии РЖД, показывающий примеры используемых понятий и отношений, показан на рис. 14.

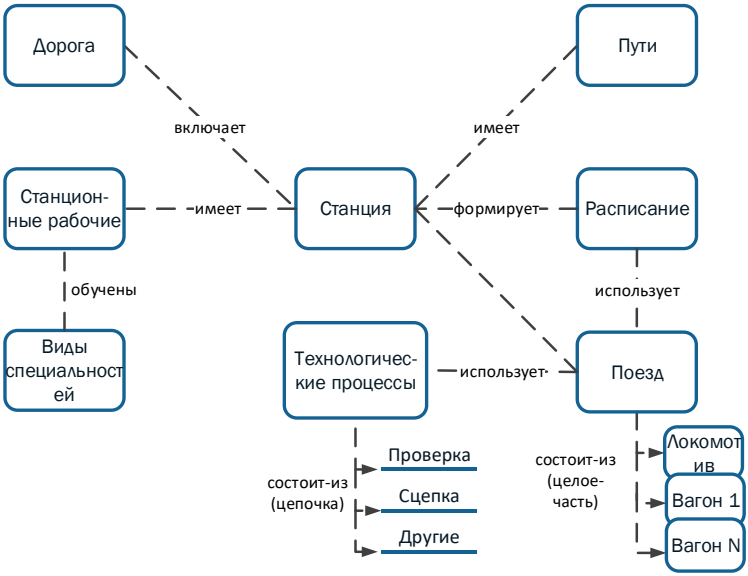

Рис. 14. Упрощенный фрагмент онтологии РЖД

Важно отметить, что выбор онтологического базиса (метаонтологии) приобретает ключевое значение в случае, когда необходимо приобретение новых знаний, например, когда в сети встречаются два агента, имеющие разные онтологии. При наличии единого базиса эти агенты имеют шанс «договориться», поскольку их системы знания базируются на единой основе. В частности, агент в ответ на

запрос с указанием на неизвестный объект сможет спросить, «что это?» и получить в качестве ответа спецификацию данного «объекта», которую мог бы встроить в свою онтологию, получив тем самым возможность рассуждать о соответствующих классах и экземплярах объектов при выработке вариантов решений.

5.3. Триада «Онтология-Модель-Сцена»

Для удобства работы с онтологиями выделяют триаду «онтология-модель-сцена» [26]. На рис. 15показана взаимосвязь между понятиями «онтология предметной области» – «модель» – «сцена».

Рис. 15. Взаимосвязь между понятиями
«онтология предметной области» – «модель» – «сцена»

Онтология описывает понятия и отношения (как толковый словарь), необходимые для описания моделей объектов любой предметной области науки (биотехнологии, наносистемы, космос,

живые системы и т.д.). Онтология включает машинно-интерпретируемые формулировки основных понятий предметной области и отношения между ними.

Преимущества применения онтологий:

- возможности и потребности объектов постоянно меняются: вводятся новые условия, меняется требования и т.д. Использование онтологий позволяет минимизировать затраты на внесение таких изменений;
- описав отдельными онтологиями (концептами и отношениями) предметные области объекта, получаем возможность использовать их для построения моделей различных составных частей системы в их конкретных конфигурациях и отражать состояние сценами («моментальными фотографиями» экземпляров объектов и отношений);
- такой подход позволит отделить знания об объектах и их связях от программного кода системы и пополнять описания мира подразделений по мере необходимости, без перепрограммирования системы в каждом случае.

Модель описывает устойчивые комбинации понятий и отношений (часть которых для удобства работы конкретизирована, например, модель продукта или технологии), упрощающие создание формализованных описаний сцен и ситуаций в конкретных задачах.

Сцена описывает экземпляры понятий и отношений в заданный момент времени (как набор фактов), необходимые для описания «фотографии» ситуации на каждом конкретной задаче с заданным уровнем подробности в каждый момент времени.

С использованием онтологии можно специфицировать конкретные факты и строить модели описания ситуаций (сцен) для работы агентов.

Сцены используются для представления и обработки оперативной информации о ситуациях, являясь «фотографией» («зерка-

лом») реальности, что позволяет системе иметь в любой момент времени формализованное описание ситуации в выбранном фрагменте реальности, используемой для принятия решений агентами.

Для представления онтологии и сцен используется семантическая сеть, которая состоит узлов и упорядоченных отношений (связей), соединяющих эти узлы. Узлы выражают понятия или предположения, а связи описывают взаимоотношения между этими узлами.

Использование онтологий, моделей и сцен нужно, прежде всего, для того, чтобы стало возможным формализовывать, специфицировать, накапливать, изменять знания, используемые при автоматизированном принятии решений. Действительно, многие объекты и связи между ними, а также списки свойств и атрибутов в процессе работы постоянно меняются и можно предложить описывать предметные области отдельными онтологиями, которые потом использовать для построения моделей в конкретных конфигурациях, и отражать состояние реальности сценами («моментальными фотографиями») объектов и отношений с нужным уровнем детализации.

Это позволит отделить знаний об объектах и их связях от программного кода системы и пополнять описания мира автоматизируемого объекта по мере необходимости, без перепрограммирования системы в случае изменений, и использовать эти знания для описания ситуаций.

5.4. Методы и средства формализованного представления онтологий

Для формализованного представления знаний наибольшее распространение получили подходы на основе фреймов, семантических сетей и продукционных правил [30]. Изначально наиболее простой формой представления онтологий были разновидности

фреймовых моделей. Но на сегодня семантические сети признаны как наиболее сложные и наиболее удобные формы концептуализации знаний. Семантические сети состоят из узлов и упорядоченных отношений (связей), соединяющих эти узлы. Узлы выражают понятия или предположения, а связи описывают взаимоотношения между этими узлами.

Для представления знаний обычно используют неоднородные семантические сети [31], в которых используют разные типы отношений. В самом простом случае неоднородную сеть можно представить в виде совокупности следующих объектов: множества S имен предметов и процессов реального мира и семейства R отношений на множестве S. Различают экстенсиональные и интенсиональные семантические сети. Экстенсиональные сети задаются посредством перечисления всех экземпляров. Интенсиональные сети – это сети, задаваемые посредством обобщающих концептов. Например, вместо отношений «Венди – дочь миссис Дарлинг» и «Майкл – сын миссис Дарлинг» будет введено отношение «быть ребенком» между детьми и родителями.

В результате развития подхода к представлению знаний в виде семантических сетей, был разработан ряд методик по их построению и использованию, в числе которых выделим динамические семантические сети (Dynamical Semantic Network (DSN)). Для DSN предлагается метод представления знаний, позволяющий строить сетевые интеллектуальные системы (или сети интеллектуальных систем). В основу концепции динамической семантической сети положены следующие принципы: интеграция процедурных и декларативных знаний; параллельное функционирование всех компонентов сети; эволюция сети в реальном времени.

Динамическая семантическая сеть – это семантическая сеть, у которой каждый узел является выполняемым вычислительным процессом, обладающим алгоритмами обработки информации и выра-

ботки решений; средствами общения с другими узлами сети; определенным поведением, что в целом очень близко понятию программного агента. На основе DSN может быть построена иерархическая DSN, представляющая собой граф, узлы которого – выполняемые процессы, обладающие набором атрибутов и присоединенных функций. Структура графа отражает текущую модель предметной области. Однако множество дуг графа не ограничивается дугами, отражающими отношения «класс-подкласс», а включает в себя дуги, отражающие отношения «система-подсистема» и ассоциативные связи между узлами. Каждый узел содержит, кроме уникальных данных, информацию о своей окрестности и множество процедур. Такая технология представляется перспективной для описания современных распределенных и эволюционирующих программных систем.

Еще один популярный способ описания онтологий – продукционные модели, которые получили распространение для описания предметных областей, в которых доминируют программные модели и алгоритмы. Однако многолетний опыт разработки экспертных систем показывает, что сложные системы правил продукций плохо поддаются формализации и структурированию и требуют постоянной перестройки и согласования.

В последнее время помимо рассмотренного формализованного «явного» представления знаний, все большее распространение получают методы их «неявного» представления, примером которого являются нейронные сети [32]. Для нейронных сетей в настоящее время также существует много развитых алгоритмов. Нейронные сети и самообучающиеся автоматы являются интересной моделью для построения самообучающихся агентов, но на практике требуют весьма сложной процедуры обучения по примерам, что в ряде случаев сделать не представляется возможным; для эффективного обучения требуется большая выборка; но самое главное, что даже на

уровне отдельных «особей» трудно выделить семантику принимаемых решений.

5.5. Методология инженерии знаний

Рассмотрим методологию создания онтологий, основанную на системах представления декларативных знаний, предложенную в [33]. Не существует единственного «правильного» способа или методологии разработки онтологий. Обсудим общие моменты, которые нужно учитывать, и рассмотрим один из возможных способов разработки онтологии. Опишем итеративный подход к разработке онтологии: начнем с первого чернового просмотра онтологии. Затем проверим и уточним получаемую онтологию и добавим детали. Попутно обсудим решения, касающиеся моделирования, которые должен принять разработчик, а также «за» и «против» и результаты принятия различных решений.

Во-первых, выделим некоторые фундаментальные правила разработки онтологии, к которым мы будем неоднократно обращаться. Эти правила могут показаться довольно категоричными. Тем не менее, во многих случаях они могут помочь принять проектные решения.

1. Не существует единственного правильного способа моделирования предметной области – всегда существуют жизнеспособные альтернативы. Лучшее решение почти всегда зависит от предполагаемого приложения и ожидаемых расширений.

2. Разработка онтологии – это обязательно итеративный процесс.

3. Понятия в онтологии должны быть близки к объектам (физическим или логическим) и отношениям в интересующей вас предметной области. Скорее всего, это существительные (объекты) или

глаголы (отношения) в предложениях, которые описывают вашу предметную область.

То есть, знание того, для чего вы собираетесь использовать онтологию и насколько детальной или общей она будет, повлияет на многие решения, касающиеся моделирования. Среди нескольких жизнеспособных альтернатив нам нужно определить, какая поможет лучше решить поставленную задачу и будет более наглядной, более расширяемой и более простой в обслуживании. Также нужно помнить, что онтология – это модель реального мира и понятия в онтологии должны отражать эту реальность. После того, как мы определим начальную версию онтологии, мы можем оценить и отладить ее, используя ее в приложениях или в методах решения задач и/или обсудив ее с экспертами предметной области. В результате почти наверняка нам нужно будет пересмотреть начальную онтологию. Этот процесс итеративного проектирования, вероятно, будет продолжаться в течение всего жизненного цикла онтологии.

Шаг 1. Определение области и масштаба онтологии

Мы предлагаем начать разработку онтологии с определения ее области и масштаба, т.е., ответим на несколько основных вопросов:

- Какую область будет охватывать онтология?
- Для чего мы собираемся использовать онтологию?
- На какие типы вопросов должна давать ответы информация в онтологии?
- Кто будет использовать и поддерживать онтологию?

Ответы на эти вопросы могут измениться во время процесса проектирования онтологии, но в любой заданный момент времени они помогают ограничить масштаб модели.

Один из способов определить масштаб онтологии – это набросать список вопросов, на которые должна ответить база знаний, основанная на онтологии, т.е. вопросы для проверки компетентности. Содержит ли онтология достаточно информации для ответа на эти

типы вопросов? Требуется ли для ответов особый уровень детализации или представление определенной области? Эти вопросы для проверки компетентности являются всего лишь формальными и не должны быть исчерпывающими.

Шаг 2. Рассмотрение вариантов повторного использования существующих онтологий

Почти всегда стоит учесть, что сделал кто-то еще, и проверить, можем ли мы улучшить и расширить существующие источники для нашей конкретной предметной области и задачи. Повторное использование существующих онтологий может быть необходимым, если нашей системе нужно взаимодействовать с другими приложениями, которые уже вошли в отдельные онтологии или контролируемые словари. Многие онтологии уже доступны в электронном виде и могут быть импортированы в используемую Вами среду проектирования онтологии. Формализм онтологии часто не имеет значения, т.к. многие системы представления знаний могут импортировать и экспортировать онтологии. Даже если система представления знаний не может работать напрямую с отдельным формализмом, задача перевода онтологии из одного формализма в другой обычно не является сложной.

В литературе и всемирной паутине существуют библиотеки повторно используемых онтологий. Например, мы можем использовать библиотеку онтологий Ontolingua (http://www.ksl.stanford.edu/software/ontolingua/) или библиотеку онтологий DAML (http://www.daml.org/ontologies/). Существует также ряд общедоступных коммерческих онтологий (например, UNSPSC (www.unspsc.org), RosettaNet (www.rosettanet.org), DMOZ (www.dmoz.org)).

Шаг 3. Перечисление важных терминов в онтологии

Полезно составить список всех терминов, о которых мы хотели бы сказать что-либо или которые хотели бы объяснить поль-

зователю. Какие термины мы бы хотели рассмотреть? Какие свойства имеют эти термины? Что бы мы хотели сказать об этих терминах?

Следующие два шага – разработка иерархии классов и определение свойств понятий (слотов) – тесно переплетены. Сложно выполнить сначала один из них, а потом – другой. Обычно в иерархии мы даем несколько формулировок понятий и затем описываем свойства этих понятий и т.д. Также эти два шага – самые важные шаги в процессе проектирования онтологии.

Шаг 4. Определение классов и иерархии классов

Существует несколько возможных подходов для разработки иерархии классов:

- Процесс нисходящей разработки начинается с определения самых общих понятий предметной области с последующей конкретизацией понятий.
- Процесс восходящей разработки начинается с определения самых конкретных классов, листьев иерархии, с последующей группировкой этих классов в более общие понятия.
- Процесс комбинированной разработки – это сочетание нисходящего и восходящего подходов: Сначала мы определяем более заметные понятия, а затем соответствующим образом обобщаем и ограничиваем их.

Ни один из этих трех методов не лучше других по своей сути. Выбор подхода в большой степени зависит от личного взгляда на предметную область. Если разработчик склонен к рассмотрению предметной области сверху вниз, то ему, возможно, больше подойдет нисходящий метод. Часто для многих разработчиков онтологий самым простым является комбинированный метод, т.к. понятия, находящиеся «посередине», имеют тенденцию быть самыми наглядными понятиями в предметной области.

Какой метод мы бы ни избрали, обычно мы начинаем с определения классов. Из списка, составленного в Шаге 3, мы выбираем термины, которые описывают объекты, существующие независимо, а не термины, которые описывают эти объекты. В онтологии эти термины будут классами и станут точками привязки в иерархии классов.

Шаг 5. Определение свойств классов – слотов

Классы сами по себе не предоставляют достаточно информации для ответа на вопросы проверки компетентности из Шага 1. После определения некоторого количества классов мы должны описать внутреннюю структуру понятий. Для каждого свойства из списка мы должны определить, какой класс оно описывает. Эти свойства станут слотами, привязанными к классам.

В онтологии слотами могут стать несколько типов свойств объектов:

- «внутренние» свойства;
- «внешние» свойства;
- части, если объект имеет структуру; они могут быть как физическими, так и абстрактными «частями»;
- отношения с другими индивидными концептами; это отношения между отдельными членами класса и другими элементами.

Шаг 6. Определение фацетов слотов

Слоты могут иметь различные фацеты, которые описывают тип значения, разрешенные значения, число значений (мощность) и другие свойства значений, которые может принимать слот.

Мощность слота определяет, сколько значений может иметь слот.

Фацет типа значения описывает, какие типы значений можно ввести в слот.

Разрешенные классы для слотов типа Экземпляр часто называют диапазоном значений слота. Классы, к которым слот привязан, или классы, свойство которых слот описывает, называются доменом слота.

При определении домена или диапазона значений слота найдите наиболее общие классы или класс, которые могут быть соответственно доменом или диапазоном значений слотов. С другой стороны, не определяйте слишком общий домен и диапазон значений: все классы в домене слота должны быть описаны слотом, а экземпляры всех классов в диапазоне значений слота должны являться потенциальными заполнителями слота. Не выбирайте слишком общий класс для диапазона значений (то есть, вы не захотите делать THING диапазоном значений, а захотите выбрать класс, который охватит все заполнители).

Шаг 7. Создание экземпляров

Последний шаг – это создание отдельных экземпляров классов в иерархии. Для определения отдельного экземпляра класса требуется (1) выбрать класс, (2) создать отдельный экземпляр этого класса и (3) ввести значения слотов.

6. ПРОГРАММНЫЕ ИНСТРУМЕНТЫ ДЛЯ РАБОТЫ С ОНТОЛОГИЯМИ

В последнее десятилетие для создания и поддержки онтологий появился целый ряд инструментов, которые помимо общих функций редактирования и просмотра выполняют поддержку документирования онтологий, импорт и экспорт онтологий разных форматов и языков, поддержку графического редактирования, управление библиотеками онтологий и другие возможности.

Наиболее известные инструменты для работы с онтологиями представлены в [34, 35].

При создании и использовании инструментов для работы с онтологиями необходимо решать следующие проблемы:

- поддержка совместных и распределенных работ, поддержка совместимости при работе на разных платформах;
- масштабируемость, расширяемость, версификация, безопасность;
- поддержка анализа, который будет фокусировать внимание пользователя на областях онтологии, где, возможно, необходимы модификации для того, чтобы сделать их более корректными;
- жизненный цикл онтологии, т.е. наличие Инструментов поддержки развития онтологий (например, инструменты слияния, система контроля кода и т.п.);
- простота использования, в том числе, поддержка простого, но разнообразного пользовательского интерфейса как для неискушенных, так и продвинутых пользователей, под-

держка разных стилей представления информации (текстовый, графический или др.).

В инженерии знаний существует понятие поля знаний, которое позволяет инженеру по знаниям описывать основные объекты предметной области и связи между ними в виде графа, диаграммы или любого другого наглядного представления. Поэтому инструменты визуального проектирования онтологий являются одной из наиболее важных частей инструментов поддержки. Преимущество визуального проектирования состоит в том, что разработчик может не заботиться о языке представления онтологии, а оперировать в терминах понятий, связей и т.п. Как следствие этого удобное графическое представление позволяет легко строить концептуальные модели сложных предметных областей даже неспециалистам. Кроме того, сам процесс визуального проектирования онтологии является «достаточно эффективным гносеологическим инструментом» [34, 35], помогая разработчику или исследователю осознавать структуру предметной области, ее основные и производные концепты и отношения (связи) между ними.

6.1. Система Ontolingua

Система Ontolingua была разработана в KSL (Knowledge Systems Laboratory) Стенфордского университета и стала первым инструментом онтологий. Она состоит из сервера и языка представления знаний [36]. Сервер Ontolingua организован в виде набора онтологий, относящихся к Web-приложениям, которые надстраиваются над системой представления знаний Ontolingua. Редактор онтологий – наиболее важное приложение сервера Ontolingua является Web-приложением на основе форм HTML. Кроме редактора

онтологий Сервер Ontolingua включает Webster (получение определений концептов), сервер OKBC (доступ к онтологиям Ontolingua по протоколу OKBC) и Chimaera (анализ, объединение, интегрирование онтологий).

Все приложения, кроме сервера OKBC, реализованы на основе форм HTML. Система представления знаний реализована на Lisp. Сервер Ontolingua также предоставляет архив онтологий, включающий большое количество онтологий различных предметных областей, что позволяет создавать онтологии из уже существующих. Сервер поддерживает совместную разработку онтологии нескольким пользователям, для чего используются понятия пользователей и групп. Система включает графический браузер, позволяющий просмотреть иерархию концептов, включая экземпляры. Ontolingua обеспечивает использование принципа множественного наследования и богатый набор примитивов. Сохраненные на сервере онтологии могут быть преобразованы в различные форматы для использования другими приложениями, а также импортированы из ряда языков в язык Ontolingua.

6.2. Система Protege

Protege – одна из наиболее известных систем на языке Java, разработанная группой медицинской информатики Стенфордского университета [22]. Программа предназначена для построения (создания, редактирования и просмотра) онтологий моделей прикладной области. Её первоначальная цель – помочь разработчикам программного обеспечения в создании и поддержке явных моделей предметной области и включение этих моделей непосредственно в программный код. Protege включает редактор онтологий, позволяющий проектировать онтологии, разворачивая иерархическую

структуру абстрактных или конкретных классов и слотов. Структура онтологии реализована аналогично иерархической структуре каталога. На основе сформированной онтологии, Protege может генерировать формы получения знаний для введения экземпляров классов и подклассов. Инструмент имеет графический интерфейс удобный для использования неопытными пользователями, снабжен справками и примерами.

Protege основан на фреймовой модели представления знания OKBC (Open Knowledge Base Connectivity) и снабжен рядом плагинов, что позволяет его адаптировать для редактирования моделей в разных форматах (стандартный текстовый, базы данных JDBC, UML, языков XML, XOL, SHOE, RDF и RDFS, DAML+OIL, OWL).

Protégé является свободной платформой с открытым исходным кодом, которая обеспечивает растущее сообщество пользователей набором инструментов для построения моделей предметных областей и приложений, основанных на знаниях, с помощью онтологий. Protégé реализует богатый набор структур и действий для моделирования знаний, которые поддерживают создание, визуализацию онтологий и манипулирование онтологиями в различных форматах представления. Protégé можно настроить для создания моделей знаний и ввода данных в конкретной предметной области. Кроме того, Protégé может быть расширен путем plug-in архитектуры и Java-based Application Programming Interface (API) для создания инструментов и приложений, основанных на знаниях.

Фрагмент интерфейса Protégé показан на

Рис 16.

Платформа Protege поддерживает два основных способа моделирования онтологий с помощью редакторов Protégé-Frames и Protégé-OWL.

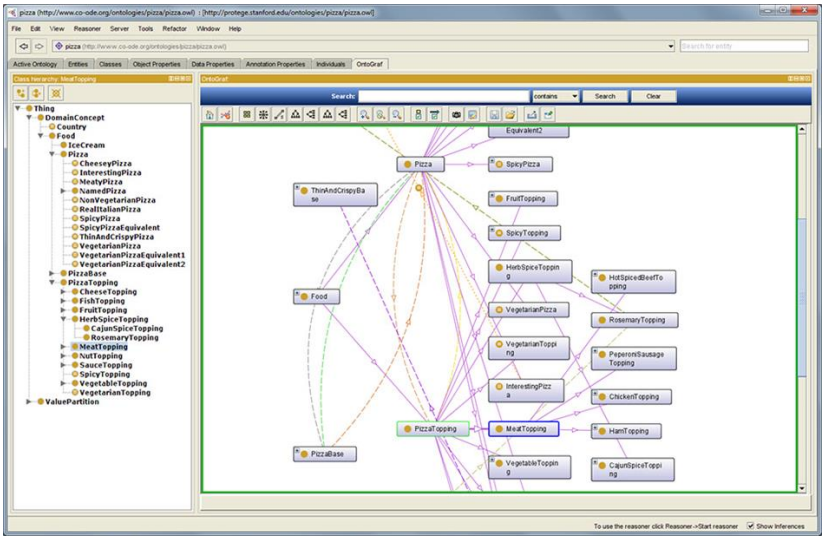

Рис. 16. Интерфейс Protege

6.3. Редактор OntoEdit

OntoEdit первоначально был разработан в институте AIFB, Университета Karlsruhe (затем коммерциализован Ontoprise GmbH) выполняет проверку, просмотр, кодирование и модификацию онтологий [37]. В настоящее время OntoEdit поддерживает языки представления: FLogic, включая машину вывода, OIL, расширение RDFS и внутреннюю, основанную на XML, сериализацию модели онтологии используя OXML. К достоинствам инструмента можноотнести удобство использования; разработку онтологии под руководством методологии и с помощью процесса логического вывода; разработку аксиом; расширяемую структуру посредством плагинов, а также очень хорошую документацию. Как и Protege, OntoEdit – автономное Java-приложение, которое можно локально установить на компьютере, но его коды закрыты.

Архитектура OntoEdit подобна Protege. Существует две версии OntoEdit: свободно распространяемая OntoEdit Free и лицензированная OntoEdit Professional. Естественно, что OntoEdit Professional имеет более широкий набор функций и возможностей (например, машину вывода, графический инструмент запросов, больше модулей экспорта и импорта, графический редактор правил, поддержка формата баз данных JDBC и т.д.).

6.4. Редактор OilEd

OilEd – автономный графический редактор онтологий, разработан в Манчестерском университете в рамках европейского 1ST проекта On-To-Knowledge [38]. Инструмент основан на языке OIL (сейчас адаптирован для DAML+OIL, в перспективе – OWL), который сочетает в себе фреймовую структуру и выразительность дескриптивной логики с сервисами рассуждения. Это позволило обеспечить понятный и интуитивный стиль интерфейса пользователя и преимущества поддержки рассуждения (обнаружение логически противоречивых классов и скрытых отношений подкласса). Из недостатков можно выделить отсутствие поддержки экземпляров.

6.5. Fluent Editor 2015

Fluent Editor 2015, редактор онтологий, является комплексным инструментом для редактирования и обработки сложных онтологий, использующим Controlled Natural Language (CNL) [39].

Fluent Editor 2015 – один из наиболее удобных для пользователей редакторов на основе OWL. Его главной особенностью является использование CNL в качестве языка моделирования знаний. С помощью Predictive Editor выполняется грамматический и морфоло-

гический анализ предложений и обеспечивается активная помощь пользователю во время написания предложения (рис. 17).

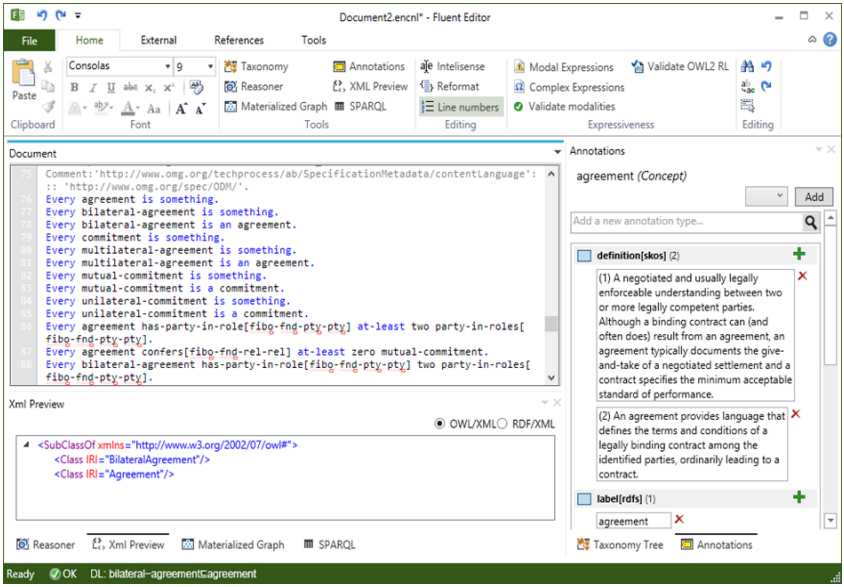

Рис. 17. Интерфейс Fluent Editor 2015

6.6. Проект CYC

Cyc представляет собой проект в области искусственного интеллекта, который пытается собрать всеобъемлющую онтологию и универсальную базу знаний с целью создания приложений, выполняющих рассуждения по типу человеческих [40].

Проект был начат в 1984 году как часть Microelectronics and Computer Technology Corporation. Цель состояла в том, чтобы закодировать в машинно- удобном формате миллионы единиц знаний людей. Cyc представила запатентованную схему представления знаний, которые использовали отношения первого порядка. В 1986

г. Cyc была наполнена 250,000 правил. Проект Cyc был выделен в Cycorp, Inc. в Остине, штат Техас в 1994 г.

Название «Cyc» является зарегистрированной торговой маркой, принадлежащей Cycorp. Первоначальная база знаний, являющаяся уменьшенной версией базы знаний, предназначенной для установления общего словаря для выполнения автоматического рассуждения, была выпущена в качестве OpenCyc по лицензии с открытым исходным кодом (Apache). Затем Cyc был сделан доступным для исследователей искусственного интеллекта под лицензией в научно-исследовательских целях как ResearchCyc.

Типичные элементы знаний, представленные в базе данных: «Каждое дерево – это растение» и «Растения в конце концов умирают». Когда построен запрос, умирают ли деревья, механизм логического вывода должен сделать очевидный вывод и ответить на вопрос правильно. База знаний содержит более одного миллиона определенных людьми утверждений, правил или общих идей. Они сформулированы на языке CYCL, который основан на исчислении предикатов и имеет синтаксис, подобный языку программирования Lisp.

Большая часть текущей работы над проектом Cyc, выполняемой в области инженерии знаний, связана с внесением фактов о мире вручную и внедрением эффективных механизмов логического вывода на этих знаниях. Разрабатываются средства для общения с конечными пользователями на естественном языке, а также для оказания помощи в процессе формирования знаний посредством машинного обучения.

Как и многие другие компании, Cycorp имеет амбиции использовать процессор обработки естественного языка, чтобы разобрать весь Интернет с целью извлечения структурированных данных.

CYC является одной из самых крупных и разработанных онтологий. Она направлена на использование в рамках искусственного

интеллекта, в ней содержится большое количество (более миллиона) аксиом. Версия ResearchCYC, преобразованная в формат RDF, не включает вторичных понятийных выражений, и таким образом, в ней пропала часть родовых понятий высшего уровня. Проблемой использования этой онтологии является запутанность ее структуры, она очень сложна для понимания.

6.7. Проект SUMO

The Suggested Upper Merged Ontology (SUMO) является онтологией верхнего уровня и задумана как фундаментальная онтология для различных систем компьютерной обработки информации. SUMO разработана Teknowledge Corporation, в настоящее время поддерживается Articulate Software. SUMO – среда с открытым исходным кодом [41].

SUMO изначально наполнялась понятиями мета-уровня (общими сущностями, которые не принадлежат к определенной предметной области), и, таким образом, естественно обеспечивала схему категоризации для энциклопедий. В настоящее время значительно расширена за счет включения онтологии среднего уровня и десятков онтологий предметных областей.

Существует ряд онтологий, которые объединяют несколько более мелких и служат своего рода каркасом для встраивания новых элементов на всех уровнях. Одной из таких онтологий является Suggested Upper Model Ontology (SUMO – (http://www.ontologyportal.org/). Она и включенные в нее онтологии предметных областей представляют собой самую большую общедоступную онтологию, существующую на сегодняшний день. SUMO состоит из 20 000 элементов и 60 000 аксиом. В нее включаются сами разработки SUMO, онтология среднего уровня (MILO) и

набор отраслевых онтологий (в сфере коммуникаций, транспорта, географии и многих других). Кроме этого, она содержит большое количество аксиом, причем все ее элементы формально определены и это описание не зависит от конкретной системы осуществления выводов.

6.8. Контрольные вопросы по разделам №5–№6

1. Каким критериям должна отвечать онтология?
2. Дайте определение модели онтологии.
3. Что такое онтологический базис? Какие виды онтологических базисов Вам известны?
4. Каковы функции метаонтологии?
5. Опишите структуру метаонтологии Аристотеля.
6. Как связаны понятия «онтология предметной области» – «модель» – «сцена»? Для чего они используются?
7. Дайте определение динамической семантической сети.
8. Объясните методологию проектирования онтологий. Какие этапы включает проектирование онтологий (перечислить).
9. Опишите сущность Этапа 1 проектирования онтологии «Определение области и масштаба онтологии». Приведите перечень вопросов для проверки компетентности для любой онтологии, кроме онтологии вин.
10. Опишите сущность Этапа 2 проектирования онтологии «Рассмотрение вариантов повторного использования существующих онтологий». Какие из существующих онтологий могут быть рассмотрены?
11. Опишите сущность Этапа 3 проектирования онтологии «Перечисление важных терминов в онтологии». Приведите примеры для любой онтологии, кроме онтологии вин.

12. Опишите сущность Этапа 4 проектирования онтологии «Определение классов и иерархии классов». Какие подходы для разработки иерархии классов Вам известны? Приведите примеры для любой онтологии, кроме онтологии вин.

13. Опишите сущность Этапа 5 проектирования онтологии «Определение свойств классов – слотов». Какие типы свойств объектов могут быть определены в качестве слотов? Приведите примеры для любой онтологии, кроме онтологии вин.

14. Опишите сущность Этапа 6 проектирования онтологии «Определение фацетов слотов». Перечислите наиболее общие фацеты. Приведите примеры для любой онтологии, кроме онтологии вин.

15. Опишите сущность Этапа 7 проектирования онтологии «Создание экземпляров». Приведите примеры для любой онтологии, кроме онтологии вин.

16. Как обеспечить правильность иерархии классов? В чем заключается различие между отношениями «is-a» и «kind-of»?

17. Что такое «инженерия онтологий»? Какие действия она выполняет?

18. Что такое программные средства работы с онтологиями? Опишите основную функциональность редакторов онтологий. Какие редакторы онтологий Вам известны?

7. ЛАБОРАТОРНАЯ РАБОТА «КОНСТРУИРОВАНИЕ ОНТОЛОГИИ С ИСПОЛЬЗОВАНИЕМ РЕДАКТОРА PROTEGE»

7.1. Цели и задачи лабораторной работы

В лабораторной работе рассматривается онтология в информационных технологиях на примере разработки простой онтологии Пиццы. В качестве инструмента разработки используется редактор онтологий Protege 4. Сначала вводится понятие онтологии предметной области, затем рассматривается процесс создания онтологии, описание свойств предметной области на языке дескриптивной логики OWL.

Цель лабораторной работы – Освоить работу с редактором онтологий Protege.

В процессе выполнения лабораторной работы № 2 решаются следующие задачи:

- освоение работы с редактором онтологий Protege;
- освоение навыков создания онтологии.

7.2. Инструкция по выполнению лабораторной работы

1. Загрузите среду Protégé [42]: http://protege.stanford.edu/download/protege/4.3/installanywhere/Web_Installers/

2. Загрузите инструкцию по разработке простой онтологии Пиццы [43]: Matthew Horridge et al. 2011. A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools, Edition 1.3,

published by the University of Manchester, 24 Mar 2011, 108 pp. – http://people.cs.vt.edu/~kafura/ComputationalThinking/Class-Notes/Tutorial-Highlighted-Day1.pdf

Русская версия инструкции изложена в [22].

3. Постройте онтологию Пиццы в соответствии с инструкцией.

7.3. Индивидуальные задания

1. Разработайте онтологию в соответствии с полученным вариантом задания:
   - создайте не менее трех уровней в дереве иерархии, начиная с класса по заданию,
   - создайте не менее пяти подклассов для каждого из созданных уровней.
2. Наполните онтологию значениями экземпляров (3-5 экземпляров).
3. Представьте результат в виде семантической сети.

7.4. Варианты заданий

1. Виды печатных изданий.
2. Виды отдыха.
3. Виды языков программирования.
4. Виды продаж.
5. Виды автомобилей.
6. Виды косметических товаров.
7. Виды литературных произведений.
8. Виды учебных заведений.
9. Виды транспорта.

10. Виды форматов файлов.
11. Виды персональных компьютеров.
12. Сорта шоколада.
13. Виды канцелярских принадлежностей.
14. Виды мобильных телефонов.
15. Виды гаджетов.
16. Виды мебели для кухни.
17. Виды цветов.
18. Виды домашних животных.
19. Виды спортивного снаряжения.
20. Виды физических величин.

7.5. Контрольные вопросы

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

ЗАКЛЮЧЕНИЕ

Онтология – это выражение человеческого познания предметной области, она оперирует понятиями, концептами и категориями, которые являются наименованием сущностей реального мира. Построить онтологию – значит построить иерархию понятий, концептов, категорий. Онтология – это терминология для обмена знаниями между исследователями в определенной предметной области.

Для потребителя информационных услуг онтология – это классификатор: разбивает объекты по группам в соответствии с определенными признаками, что позволяет одну и ту же информацию организовывать различными способами в соответствии с потребностями пользователя. Эта возможность полезна при разработке приложений, работающих с базами данных.

В то же время, онтология – это средство общения, так как класс обобщает все подчиненные ему концепты, что может быть использовано при информационном поиске.

Онтология дает возможность получать путем вывода новые знания, в сокращенном виде передает смысл информации, может из разных представлений выделять общее необходимое потребителю знание.

Изнутри онтология – это база знаний, включающая утверждения общего характера о предметной области и утверждения частного характера об индивидуальностях. Все утверждения выражены в формализме дескрипционной логики и записаны на языке OWL. Это описание выполняется автоматически с использованием редакторов. Язык OWL может быть однозначно прочитан и понят в любой точке всемирной сети и при использовании соответствующих

средств – парсеров информация об онтологии будет представлена в понятном для пользователя графическом виде (в виде семантической сети или дерева концептов). Дескрипционная логика позволяет осуществлять вывод не заложенных в явном виде знаний, которые также предоставляются пользователю. Онтология, записанная на языке OWL, понятна программным агентам и может быть использована ими для обмена знаниями.

Не существует единственной правильной онтологии для какой-либо предметной области. Разработка онтологии – творческий процесс, поэтому две онтологии, разработанные разными людьми, не будут одинаковыми. Потенциальные применения онтологии и точка зрения разработчика на предметную область влияют на выбор структуры онтологии.

СПИСОК ЛИТЕРАТУРЫ

1. Building Semantic Networks from Plain Text and Wikipedia with Application to Semantic Relatedness and Noun Compound Paraphrasing / P.-R. Wojtinnek, S. Pulman // International J. of Semantic Computing. – 2012. Volume 06. Number 01. DOI:10.1142/S1793351X12400041.
2. The Generic Frame Protocol / P. Karp, K. Myers, T. Gruber // Proc. 14th Int. Joint Conf. on Artificial Intelligence, Montréal Québec, Canada, Aug. 20–25. – 1995. – P. 768–774.
3. The Logic Model: A Tool for Incorporating Theory in Development and Evaluation of Programs // R. Savaya, M. Waysman // Administration in Social Work. – 2005. – Volume 29. Number 2. – P. 85–103.
4. Vadapalli, P. Production System in Artificial Intelligence and its Characteristics // Product Management in AI: [сайт]. – 2022. – URL: https://www.upgrad.com/blog/production-system-in-artificial-intelligence/ (дата обращения: 25.01.2022).
5. Gruber, T. The role of common ontology in achieving sharable, reusable knowledge bases // In J. Allen, R. Fikes, and E. Sandewell (Eds). Principles of Knowledge Representation and Reasoning. – Morgan Kaufmann, 1991. – P. 601-602.
6. Uschold, M., Gruninger, M. Ontologies: Principles, methods and applications // Knowledge Engineering Review. – 1996. Volume 11. Number.2. – P. 93-155.
7. Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. A New Form of Web Content // Scientific American. – 2001. – Volume 284. – P. 1–5.
8. Gruber, T. Ontology // L. Liu, M. Tamer Özsu (Eds.). Encyclopedia of Database Systems. – Springer-Verlag, 2009. – P. 1963–1965.

9. Добров, Б.В. Онтологии и тезаурусы: модели, инструменты, приложения / Б.В. Добров, В.В. Иванов, Н.В. Лукашевич, В.Д. Соловьев // Интернет-университет: [сайт]. – 2022. – UR: http://www.intuit.ru/department/expert/ontoth/ (дата обращения: 25.01.2022).
10. OpenCyc: [сайт]. – 2022. – URL: https://www.ime.usp.br/~fr/opencyc/ (дата обращения: 03.02.2022).
11. DOLCE: Descriptive Ontology for Linguistic and Cognitive Engineering: [сайт]. – 2022. – URL: http://www.loa.istc.cnr.it/dolce/overview.html (дата обращения: 04.02.2022).
12. Suggested Upper Merged Ontology (SUMO): [сайт]. – 2022. – URL: http://ontologyportal.org/ (дата обращения: 05.02.2022).
13. DAML Language: [сайт]. – 2022. – URL: http://www.daml.org/about.html (дата обращения: 06.02.2022).
14. Ontology Inference Layer (OIL): [сайт]. – 2022. – URL: http://xml.coverpages.org/oil.html (дата обращения: 06.02.2022).
15. M. Ribière, P. Charlton. Ontology Overview / Motorola Labs, Paris: [сайт]. – 2022. – URL: http://www.fipa.org/docs/input/f-in-00045/f-in-00045.pdf (дата обращения: 07.02.2022).
16. Resource Description Framework (RDF): [сайт]. – 2022. – URL: https://www.w3.org/RDF/ (дата обращения: 08.02.2022).
17. Бездушный А.А. Архитектура RDFS-системы. Практика использования открытых стандартов и технологий Semantic Web в системе ИСИР / Бездушный А.А., Бездушный А.Н., Нестеренко А.К. [и др.]: [сайт]. – 2003. – URL: http://rcdl.ru/doc/2003/J1.pdf (дата обращения: 09.02.2022).
18. Dublin Core Activity: [сайт]. – 2022. – URL: http://dublincore.org (дата обращения: 10.02.2022).
19. Web Ontology Language (OWL) / OWL Working Group: [сайт]. – 2022. – URL: https://www.w3.org/2001/sw/wiki/OWL (дата обращения: 10.02.2022).

20. Лапшин, В.А. Онтологии в компьютерных системах / В.А. Лапшин. – Москва: Научный мир, 2010. – 224 с.
21. Гаврилова, Т.А. Базы знаний интеллектуальных систем: Учебник для вузов / Т.А. Гаврилова, В.Ф. Хорошевский. – Санкт-Петербург: Питер, 2001. – 384 с.
22. Цуканова, Н.И. Онтологическая модель представления и организации знаний: учебное пособие для вузов / Н.И. Цуканова. – Мочква: Горячая линия – Телеком, 2015. – 272 с.
23. Gruber, T. Toward principles for the design of ontologies used for knowledge sharing? // International Journal of Human-Computer Studies. – 1995. Volume 43. Issues 5-6. – P. 907–928.
24. Gomez-Perez, A., Benjamins, V. Overview of Knowledge Sharing and Reuse Components: Ontologies and Problem-Solving Methods: [сайт]. – 2022. – URL: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.39.249&rep=rep1&type=pdf (дата обращения: 10.02.2022).
25. Guarino, N., Welty, Ch. A Formal Ontology of Properties: [сайт]. – 2022. – URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.7392&rep=rep1&type=pdf (дата обращения: 11.02.2022).
26. Skobelev, P., Activity ontology for situational management of enterprises in real time // Ontology of designing. – 2012. – Volume 1. Number 3. – P. 6–38.
27. Altuner, İ. Ontological Bases of the Universe in Plato's and Aristotle's Cosmologies // Journal of Social Sciences. – 2013. – Number 3. – P. 1–10.
28. Скобелев, П.О. Применение онтологии в интеллектуальной системе распределенного управления группировкой малоразмерных космических аппаратов / П.О. Скобелев, Е.В. Симонова, М.Е. Степанов [и др.] // Известия Самарского научного центра

Российской академии наук. – 2015. – Том 17, №2(5). – С. 1119–1130.

29. Шабунин, А.Б. Разработка онтологии для мультиагентной системы управления ресурсами ОАО «РЖД» / А.Б. Шабунин, Н.А. Кузнецов, П.О. Скобелев [и др.] // Информационные технологии. – 2012. – № 12. – С. 42–45.
30. Гаврилова, Т.А. Инженерия знаний. Модели и методы: учебник для вузов / Т.А. Гаврилова, Д.В. Кудрявцев, Д.И. Муромцев. – Санкт-Петербург: Лань, 2022. – 324 с.
31. Huhns, M., Singh, M. Ontologies for Agents // IEEE Internet Computing. – 1997. – November – December. – P. 17–24.
32. Осипов, Г.С. Приобретение знаний интеллектуальными системами: Основы теории и технологии / Г.С. Осипов. – Москва: Наука, 1997. – 112 с.
33. Noy, N., McGuinness, D. Ontology Development 101: A Guide to Creating Your First Ontology. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, March 2001: [сайт]. – 2022. – URL: https://protege.stanford.edu/publications/ontology_development/ontology101.pdf (дата обращения: 11.02.2022).
34. Константинова, Н.С. Онтологии как системы хранения знаний / Н.С. Константинова, О.А. Митрофанова. – 2008: [сайт]. – 2022. – URL: http://window.edu.ru/resource/795/58795 (дата обращения: 12.02.2022).
35. Овдей, О.М. Обзор инструментов инженерии онтологий / О.М. Овдей, Г.Ю. Проскудина // Российские цифровые библиотеки. – 2004. – Том 7, Номер 4: [сайт]. – 2022. – URL: http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2004/part4/op/ (дата обращения: 13.02.2022).
36. Farquhar, A., Fikes, R., Rice, J. The ontolingua server: A tool for collaborative ontology construction // International Journal of Human-Computer Studies. – 1997. – Volume 46. Number 6. – P. 707–728.

37. Sure-Vetter, Y., Erdmann, M. OntoEdit: Collaborative Ontology Development for the Semantic Web // LNCS. – 2002. – Volume 2342. – P. 221–235.
38. Bechhofer, S., Horrocks, I., Goble C. OilEd: A Reason-able Ontology Editor for the Semantic Web // LNAI. – 2001. – Volume 2174. – P.396-408.
39. Fluent Editor 2015: [сайт]. – 2022. – URL: http://www.cognitum.eu/semantics/FluentEditor/ (дата обращения: 14.02.2022).
40. Matuszek, C, Witbrock, M, Kahlert, R. Lenat. Searching for Common Sense: Populating Cyc from the Web: [сайт]. – 2022. – URL: https://www.aaai.org/Papers/AAAI/2005/AAAI05-227.pdf (дата обращения: 15.02.2022).
41. Niles, I. Mapping WordNet to the SUMO ontology: [сайт]. – 2022. – URL: https://www.researchgate.net/publication/228732092_ Mapping_WordNet_to_the_SUMO_ontology (дата обращения: 16.02.2022).
42. Protégé: [сайт]. – 2022. – URL: http://protege.stanford.edu/download/protege/4.3/installanywhere/Web_Installers/ (дата обращения: 16.02.2022).
43. Horridge, M. A Practical Guide To Building OWL Ontologies Using Protege 4 and CO-ODE Tools, Edition 1.3, published by the University of Manchester, 2011: [сайт]. – 2022. – URL: http://people.cs.vt.edu/~kafura/ComputationalThinking/Class-Notes/Tutorial-Highlighted-Day1.pdf (дата обращения: 17.02.2022).

Учебное издание

Симонова Елена Витальевна

МОДЕЛИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ. Часть II. Использование онтологии для моделирования сложных адаптивных систем

Учебное пособие

Редакционно-издательская обработка Л.Р. Дмитриенко
Компьютерная верстка Л.Р. Дмитриенко

Подписано в печать 28.06.2022. Формат 60x84 1/16.
Бумага офсетная. Печ. л. 7,0.
Тираж 25 экз. Заказ . Арт. – 21(Р1У)/2022.

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, Самара, Московское шоссе, 34.

Издательство Самарского университета.
443086, Самара, Московское шоссе, 34.