Онтологии в компьютерных системах

Лапшин Владимир Анатольевич, к. ф-м. н., КФМН, РГГУ, mefrill@yandex.ru

Роль онтологий в современной компьютерной науке

Уровень сложности: ★★☆

В статье обсуждаются вопросы, связанные с ролью онтологий в современных компьютерных системах. Рассматриваются различные аспекты применения онтологий для улучшения качества информационного поиска, а также для систематизации. Рассмотрены принципы, лежащие в основе языков описания семантики Web RDF и OWL. В статье также дано описание принципов, лежащих в основе языков RDF и OWL.

Требуется знание C#.

Ключевые слова: Онтологии; RDF; OWL

Что такое онтология

Мотивация

Хотя термин «онтология» сейчас достаточно популярен в программистском сообществе, четкого его понимания еще не сложилось. Знания о том, что такое онтологии, и как их использовать при создании информационных систем, до сих пор являются чем-то эзотерическим, доступным только избранным специалистам по обработке знаний. Другое мнение состоит в том, что онтологии представляют собой нечто абстрактное, неприменимое на практике «игрушечное знание», которым занимаются в своих «отвлеченных сферах» так называемые «crazy scientists», в просторечье именуемые «ботанами». Между тем, термин «онтология» совсем не сложен для понимания и был придуман для достижения вполне практических целей. В этой статье автор постарается объяснить, для чего были придуманы онтологии, и как их можно использовать при построении информационных систем.

Конечно, представить в журнальной публикации подробную онтологию использования онтологий в компьютерных системах не представляется возможным. Для полноценного описания всех аспектов такого использования необходима целая книга. В связи с этим автор рад представить свою книгу «Онтологии в компьютерных системах», изданную в 2010 году в издательстве «Научный мир» [1].

Онтологии, упрощенно говоря, представляют собой описания знаний, сделанные достаточно формально, чтобы быть обработаны компьютерами. Такие формальные описания используются в самых различных и порой достаточно неожиданных областях компьютерной науки. Далее мы рассмотрим, какие обстоятельства привели к возникновению термина «онтология», а также опишем некоторые популярные аспекты его использования при написании программ.

Онтологии как интерфейсы интеллектуальных систем

Термин «онтология» впервые появился в работе Томаса Грубера [2], в которой рассматривались различные аспекты взаимодействия интеллектуальных систем между собой и с человеком. Интеллектуальными системами называются программы, которые моделируют некоторые аспекты интеллектуальной деятельности человека. Конечно, любая программа занимается таким моделированием в той или иной степени, ведь именно в этом и состоит ценность компьютера для человека — компьютерная система позволяет освободить человека от выполнения какой-то однотипной деятельности. Эта деятельность может быть весьма сложной и изощренной, но она всегда однотипна: компьютерная система, созданная, например, для редактирования графики, не может быть использована для управления комбайнами во время сенокоса. В этом смысле знания, которые закладывает в программу ее создатель (т.е. алгоритм этой программы), всегда статичны, они не меняются (конечно, за исключением очень конкретных знаний, которые мы называем «данными программы»). Интеллектуальная система в этом смысле более универсальна — в ней знание о том, что надо делать в процессе исполнения программы, не вшито в программу раз и навсегда, но может меняться. Если так, то эти знания необходимо передавать программе как данные, т.е. возникает необходимость их описания.

Знания, которые заложены в компьютерных программах, можно разделить на два сорта:

1. *Процедурные* знания, т.е. знания о том, что надо сделать в каждой конкретной ситуации. Например, если бухгалтерской программе пришли данные о платежах, то надо сделать соответствующие изменения на счетах получателей платежей, а также другие необходимые действия, налагаемые данной ситуацией.
2. Кроме процедурных знаний, каждой программе необходимы знания *о мире задачи* или *декларативные* знания, т.е. о том, что такое платежи, проводки, счета и т.п. вещи. Без этих знаний, очевидно, программа не сможет функционировать, нельзя будет построить алгоритм программной системы.

Таким образом, при создании интеллектуальной системы приходится учитывать такое разделение знаний и придумывать какие-то программные инструменты для оперирования этими знаниями.

Томас Грубер рассматривал вопросы взаимодействия интеллектуальных систем между собой, а также с человеком. Идея Грубера состояла в том, чтобы позволить интеллектуальным системам обмениваться между собой заложенными в них знаниями о мирах задач. Если внутри интеллектуальной системы знания о мире могут быть закодированы как угодно, то для обмена этими знаниями с другой интеллектуальной системой необходимо предоставить *описание* этих знаний. Это описание должно быть в достаточной степени формальным, чтобы быть понятным другой системе, а также должен быть известен язык этого описания. Кроме того, описание должно быть понятно также и человеку. Для этого Грубер предложил описывать знания двумя способами:

- В канонической форме, которая представляет собой описание знаний на языке логики предикатов (например, в виде фактов языка Prolog).
- В форме *онтологии*, которая представляет собой множество классов, связанных между собой отношением обобщения (это обратное отношение для отношения наследования).

Таким образом, онтология по Груберу представляет собой описание декларативных знаний, сделанное в виде классов с отношением иерархии между ними. К этому описанию, предназначенному для чтения человеком, присоединено описание в канонической форме, которое предназначено для чтения машинами. Каждая интеллектуальная система может предоставлять несколько таких описаний, соответствующих различным областям хранящихся в ней декларативных знаний и, таким образом, выступает как хранилище *библиотеки* онтологий. Грубер представлял, что интеллектуальные системы будут выступать как библиотеки онтологий и свободно обмениваться онтологиями между собой. При этом библиотеке онтологий уже не обязательно быть интеллектуальной системой, достаточно просто предоставлять сервис по передаче онтологий по требованию.

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

Современное понимание термина «онтология»

Введенное Грубером разделение спецификаций знаний на две составляющие (каноническую форму и онтологию) не очень удобно, т.к. приходится описывать одни и те же знания два раза. Современные языки описания онтологий позволяют совместить эти формы спецификаций в единое целое. Таким образом, сейчас под онтологией понимается любое описание декларативных знаний, сделанное на формальном языке и снабженное некоторой классификацией специфицируемых знаний, позволяющей человеку удобно воспринимать их. Каноническая форма не обязательно использует язык логики предикатов, могут использоваться и другие формализмы. Например, можно использовать т.н. алгебраический подход к описанию знаний [3], в котором факты представляются в виде термов, а различные соотношения между фактами — в виде ограничений, налагаемых на вид этих фактов, и выраженных в форме аксиом эквивалентности. Но любое такое описание должно включать в себя представление декларативных знаний в виде иерархии объектов (классов), только в этом случае это описание может считаться онтологией.

Некоторые авторы (см. например [4,5]) используют термин «онтология» только для спецификаций знаний о мире, т.е. таких концептуализаций, целью которых является описание структуры Бытия безотносительно какой-либо инженерной задачи. Такой концептуализацией уже давно занимаются философы, и в философии термин «онтология» применяется именно в этом смысле — как спецификация знаний об окружающем мире. Программисты, однако, сталкиваются с иного рода задачами: они проводят концептуализацию с целью построения модели решаемой задачи. Таким образом, философы и программисты преследуют различные цели, когда проводят концептуализацию: первые имеют целью описание свойств окружающей реальности, а вторые строят формальную модель конкретной задачи. Для философских концептуализаций предлагается использовать термин «онтология», а для концептуализаций инженерных задач — термин «концептуальная схема». Последний термин уже используется в теории баз данных, где обозначает результат построения модели задачи. В реляционных базах данных концептуальная схема представляет собой не что иное, как схему базы данных. Термин «концептуальная схема», который предлагается использовать для обозначения спецификаций концептуализаций программных моделей, представляет собой более широкое понятие. В данной статье не проводится такого разделения и используем термин «онтология» для обозначения любого описания знаний, безотносительно того, с какими целями это описание проводилось.

Читатель может задаться вопросом: для каких целей вообще может понадобиться построение онтологий в философском смысле, кроме какой-нибудь специфической задачи построения онтологии для некоторой программы философского справочника? Оказывается, спецификации такого рода необходимы для задачи слияния онтологий. Для слияния онтологий необходимо, чтобы эти онтологии были каким-то образом согласованы друг с другом: должна быть согласована терминология, выделены термины, обозначающие одни и те же классы, описываемые факты не должны противоречить друг другу. Только в этом случае можно попытаться слить две разных спецификации в одну. Здесь на помощь приходит философская онтология. Сливаемые онтологии сначала присоединяются к философской онтологии, и на основе этого присоединения происходит согласование сливаемых онтологий. Философские онтологии в абсолютном большинстве случаев описывают очень абстрактные знания, без какой-либо конкретики, поэтому их часто называют онтологиями *верхнего уровня*. Например, в онтологии верхнего уровня придется описать, что такое материальный объект, чем он отличается от нематериального, и тому подобные вещи. Сейчас имеется довольно большое число онтологий верхнего уровня, использующих различные подходы для концептуализации Бытия, с их описанием заинтересованный читатель может ознакомиться в [1].

Онтологии в Интернет

До сих пор повествование было посвящено довольно абстрактным вещам, настало время перейти к чему-то более конкретному. В данном разделе мы рассмотрим, как онтологии применяются для описания содержимого Web-страниц.

Зачем нужно описывать содержимое Web-страницы

Онтологии содержимого Web-страниц необходимы поисковым программам для улучшения качества поиска по Web. Идея построения спецификаций концептуализаций содержания Web-страниц находится в основании концепции так называемого *Умного* Web или *Semantic Web*. Semantic Web представляет собой следующее поколение World Wide Web, в котором кроме гипертекстовых документов содержатся описания семантики этих документов, а также описания семантики различных сервисов, предоставляющих эти документы конечным пользователям. Обычно о Semantic Web рассказывают как о компоненте грядущей версии Web — так называемом Web 3.0. Каким на самом деле будет Web 3.0, мы можем только предполагать, но очевидно, что одним из главных его компонентов будет Semantic Web, в котором каждая Web-страница предоставляет также онтологию своего содержимого.

Примечание:

Термин «Web 3.0» является производным от «Web 2.0». Этим термином обозначают текущее состояние Web, которое принципиально отличается от того, в котором Web находился при его зарождении. С самого начала Web задумывался как распределенное по всему миру хранилище гипертекстовых документов, таковым Web остается и сегодня. Но с начала нынешнего века, с ростом числа пользователей, Web превратился в социальный феномен. Сегодня популярными сервисами являются не только те, которые предоставляют информацию, но и те, которые просто обеспечивают общение пользователей друг с другом. Такие сервисы получили название «социальные сети». Социальная сеть — совершенно новый феномен, который отличается от сети обычных гипертекстовых документов, как по принципам использования, так и по идеологии. Web социальных сетей — это Web 2.0. Одним из наиболее существенных отличий Web 2.0 от Web предыдущего поколения является то, что содержимое Web-страниц формируется пользователями, тогда как в старом Web это была задача разработчиков хранилищ документов — сайтов, на которых находились Web-страницы.

Формальная спецификация содержимого Web-документа дает возможность поисковой программе делать выводы о соответствии поискового запроса данному Web-документу не только на основе синтаксической информации, получаемой из текста этого документа, но и основываясь на семантике содержания данного документа. Это может кардинально улучшить качество Web-поиска, так как описание мира Web-страницы, понятное поисковой программе, дает последней гораздо больше информации, чем она может получить из неструктурированного текста.

Идеи умного Web давно были восприняты сообществом W3, в результате чего уже на протяжении более десяти лет ведутся работы по воплощению этих идей в жизнь. Первой задачей, которую необходимо решить для этого, является разработка стандартного языка, который был бы понятен всем поисковым программам. На настоящий момент разработаны два таких языка:

- Язык Resource Description Framework (RDF) — система описания ресурсов Web.
- Web Ontology Language (OWL) — язык онтологии Web. OWL можно рассматривать как расширение языка RDF.
- В следующих подразделах будут представлены описания этих языков.

Язык RDF

Язык RDF [6] разработан для того, чтобы описывать содержимое Web. В Semantic Web, когда говорят о каких-то сущностях Web, называют эти сущности ресурсами. RDF представляет собой язык для описания таких ресурсов. Ввиду того что описания семантики документов должны быть понятны компьютерам, необходимо разработать специальные программы-агенты, которые производили бы такое чтение. Также необходимо обеспечить возможность обмена информацией между различными программными агентами. Таким образом, под RDF подразумевается не только сам язык, но также и различные дополнительные программные модули, необходимые для обеспечения полноценного чтения и обмена информацией, записанной на этом языке. Этот факт подчеркивается в названии языка RDF.

Главный элемент языка RDF—это тройка, или триплет. Тройка представляет собой совокупность трех сущностей:

1. Субъект.
2. Объект.
3. Предикат.

Предикаты еще часто называют свойствами. Тройка имеет также представление в виде графа вида субъект—предикат—объект, где субъект и объект представлены как узлы, а предикат выступает в роли ребра, которое эти узлы соединяет.

С математической точки зрения, тройка представляет собой экземпляр некоторого бинарного отношения. Отношение — это множество последовательностей, состоящих в точности из n элементов для некоторого заранее определенного натурального числа n. Если n = 2, то отношение называется бинарным, т.е. бинарное отношение представляет собой множество пар. Например, отношение «семейные пары» задает множество пар элементов вида («муж», «жена»). Каждая тройка определяет одну пару из некоторого бинарного отношения, но кроме этого дополнительно задает еще имя отношения, т.е. если имеется пара («муж», «жена») отношения «семейные пары», то эту пару можно выразить тройкой («муж», «семейные пары», «жена»). Для полноценного описания, кроме задания отношений, необходимо еще задать ограничения на их содержимое. Например, для отношения «семейные пары» необходимо уточнить, что первый элемент каждой пары этого отношения должен быть мужчиной, а второй — женщиной. Для задания такого ограничения необходимо ввести понятия «мужчина» и «женщина», после чего задать ограничение, выражающее тот факт, что если имеется тройка вида («имя 1», «семейные пары», «имя 2»), то сущность «имя 1» должна быть экземпляром понятия «мужчина», а сущность «имя 2» — экземпляром понятия «женщина». Это делается посредством т.н. *высказываний*. Для высказываний существует свой язык, включающий переменные и логические операции. В качестве логических операций могут выступать: логическое «или» (дизъюнкция), логическое «и» (конъюнкция) и логическое следование (импликация). Имеются также кванторы существования и всеобщности, позволяющие ограничивать область применения высказывания. Язык RDF основан на математическом аппарате дескрипционной логики [7].

Дескрипционная логика (Description Logic — DL) базируется на формализмах семантических сетей [8] и фреймов [9], но использует аппарат математической логики. В математической логике производится явное разделение на синтаксис и семантику. Синтаксис задает язык, с помощью которого записываются различные высказывания об элемен-

тах мира данной логической системы. Семантика задает ту часть описываемого мира, которая удовлетворяет заданным ограничениям. Таких частей может быть более одной или даже бесконечно много. Каждая такая часть мира называется *моделью* данной логической системы. Опишем ограничения, налагаемые на синтаксис и семантику дескрипционных логик.

Синтаксис

Язык любой дескрипционной логики состоит из следующих элементов:

- Множество унарных предикатных символов, обозначающих имена понятий.
- Множество бинарных предикатных символов, обозначающих имена ролей.
- Рекурсивное определение термов понятий, задаваемое с помощью конструкторов на основе понятий и ролей.
- Понятия обозначают множества сущностей, которые им принадлежат, т.е. это классы в программистской терминологии. Роли задают отношения между понятиями. В качестве конструкторов термов выступают как операции логики первого порядка, такие, как перечисленные выше конъюнкция, дизъюнкция, ограничения универсальности и существования и т.д., так и операции, задающие ограничения ролей, т.е. бинарных отношений.

Ввиду того, что RDF предполагается использовать для описания ресурсов, распределенных по разным участкам Web, необходимо как-то решить проблему идентификации имен узлов и ребер RDF графа, т.е. элементов троек. Для этого используется стандартный подход: каждый элемент описывается посредством так называемого Унифицированного Идентификатора Ресурса (URI — Uniform Resource Identifer [10]). Обычно URI представляет собой либо URL (Унифицированный Указатель Ресурса—Uniform Resource Locator [11]), содержащий информацию о местонахождении данного ресурса в Web, либо URN (Унифицированное Имя Ресурса — Uniform Resource Name [12]), позволяющий идентифицировать данный ресурс в некотором пространстве имен. Пространство имен представляет собой просто именованное множество элементов и используется, чтобы обеспечить уникальность имен этих элементов в Web.

В Semantic Web используются три стандартных пространства имен:

- rdf. В этом пространстве имен задаются имена, которые используются в RDF. URI для этого имени: http://www.w3.org/1999/02/22-rdf-syntax-ns#.
- rdfs. Здесь задаются имена, используемые в RDF Schema. Его URI: http://www.w3.org/2000/01/rdf-schema#.
- owl. Описываются имена, используемые в OWL. URI для этого имени: http://www.w3.org/2002/07/owl#.

Для описания троек в языке RDF чаще всего используются две нотации:

- Нотация N-Triple [13]. В данной нотации каждый элемент тройки представляется посредством заключенного в угловые скобки URI. Элементы идут в порядке «субъект, предикат, объект» и заканчиваются точкой. Примеры этой нотации приведены ниже.
- Нотация, основанная на языке XML. Так как консорциум Web принял язык XML в качестве основного языка, то эта нотация использует наиболее часто.

Семантика

В математической логике в качестве модели логической системы обычно используется некоторое множество, назовем его M. Каждому элементу множества унарных предикатных символов дается в соответствие некоторое унарное отношение на этом множестве, а каждому элементу множества бинарных предикатных символов — бинарное отношение. Иначе говоря, каждому имени понятия соответствует некий класс-подмножество множества M, а каждому имени роли — бинарное отношение на множестве M. Если задаются какие-то ограничения на отношения, то они должны выполняться на всех отношениях. Каждое такое задание соответствия между именами и отношениями на множестве M (обозначим его через I) будем называть моделью данной дескрипционной логики, или ее интерпретацией. Интерпретация логической системы в некотором множестве представляет собой классический подход, но в RDF используется также интерпретация на графе. Иначе говоря, в качестве моделей может выступать граф, представляющий собой описанные выше тройки. Подробности об интерпретациях на графах можно узнать из документа [14].

В дескрипционных логиках проводится различие между т.н. терминологическим компонентом — TBox (terminological box) и компонентом суждений — ABox (assertional box). TBox содержит высказывания, касающиеся иерархии понятий, т.е. задающие отношения между понятиями, а ABox содержит высказывания, характеризующие отношения индивидов и понятий. Например, высказывание «каждый пользователь — это человек» задает отношение между понятиями «пользователь» и «человек», следовательно, принадлежит множеству TBox. Высказывание «Иван является пользователем» задает отношение между индивидом «Иван» и понятием «человек» и принадлежит множеству ABox. В дескрипционной логике элементы множества TBox представляют собой ограничения, задаваемые исключительно унарными предикатами (понятиями), в этом состоит их отличие от высказываний множества ABox. Различение высказываний на TBox и ABox полезно, если рассматривать возможность построения процедуры логического вывода на моделях дескрипционных логик. Высказывания из TBox задают свойства «классификации», а высказывания из ABox — свойства, которые можно условно назвать свойствами «проверки экземпляра». Логический вывод по этим множествам может существенно различаться по производительности, поэтому имеет смысл реализовать отдельные алгоритмы вывода для каждого компонента.

Имеется множество стандартных видов дескрипционной логики, задаваемых различными ограничениями на виды отношений, которые в этих видах логики могут быть заданы. Здесь мы их перечислять не будем, обратимся лучше к конкретному примеру.

Пример спецификации онтологии на RDF

Предположим, что имеется некая база данных, содержащая информацию о пользователях. Информация о пользователе содержит следующие поля:

- Идентификатор (id).
- Имя (fname).
- Фамилия (sname).
- Возраст (age).
- Пол (sex).

База данных содержит информацию о трех пользователях:

id	Имя	Фамилия	Возраст	Пол
1	Иван	Брусенко	39	мужской
2	Шамиль	Галимов	37	мужской
3	Татьяна	Волкова	23	женский

Рассмотрим, как эти данные можно описать в виде онтологии на языке RDF. Принцип здесь простой: каждая строка таблицы представляется как экземпляр класса user. Элементы соответствующих столбцов в этом случае выступают как поля класса, что выражается в языке RDF в виде т.н. *свойств* — экземпляров класса rdf:Property. Чтобы выразить на языке RDF этот факт, будет использоваться специ-

альный предикат rdf:type. Итак, имеем объявления полей как специальных свойств:

```
(usr:id, rdf:type, rdf:Property)
(usr:fname, rdf:type, rdf:Property)
(usr:sname, rdf:type, rdf:Property)
(usr:age, rdf:type, rdf:Property)
(usr:sex, rdf:type, rdf:Property)
```

Примечание:

Обратите внимание, что все имена полей объявлены в собственном пространстве имен usr.

Рассмотрим теперь, как задается таблица данных. Сначала для каждой строки таблицы зададим экземпляр класса «user»:

```
(usr:user1, rdf:type, usr:user)
(usr:user2, rdf:type, usr:user)
(usr:user3, rdf:type, usr:user)
```

Наконец, можно приступить к описанию самих данных:

```
(usr:user1,usr:id,1)
(usr:user1,usr:fname,Иван)
(usr:user1,usr:sname,Брусенко)
(usr:user1,usr:age,39)
(usr:user1,usr:sex,Мужской)
(usr:user2,usr:id,2)
(usr:user2,usr:fname,Шамиль)
(usr:user2,usr:sname,Галимов)
(usr:user2,usr:age,37)
(usr:user2,usr:sex,Мужской)
(usr:user3,usr:id,3)
(usr:user3,usr:fname,Татьяна)
(usr:user3,usr:sname,Волкова)
(usr:user3,usr:age,23)
(usr:user3,usr:sex,Женский)
```

Схема RDF

Схема RDF (RDF Schema, RDFS [15]) представляет собой расширение языка RDF, позволяющее описывать простые онтологии данных, находящихся в хранилищах RDF. Так же, как схема базы данных описывает структуру базы данных в виде заголовков таблиц и связей между ними, схема RDF позволяет описывать структуру RDF-хранилища. Структура описывает хранилище в терминах типов и отношений между ними. На самом деле, как в этом чуть позже убедится читатель, схема RDF позволяет описывать только классификации с некоторыми дополнительными отношениями. Чтобы описать более сложные виды отношений, необходимо привлекать более мощные средства, такие, как OWL.

В RDFS можно задавать классы, которые определяются в дескриптивной логике как унарные отношения. Для этого в RDFS определен специальный объект rdfs:Class — класс всех классов. Вообще, каждый объект RDF — это экземпляр класса rdfs:Resource, и rdfs:Class здесь не исключение. Но, с другой стороны, rdfs:Resource — это класс, а значит должен быть определен как экземпляр объекта rdfs:Class. Таким образом, объекты rdfs:Resource, и rdfs:Class определяются рекурсивно посредством друг друга — случай нередкий в языках описания онтологий.

Как уже говорилось выше, сказать, что данный объект является экземпляром данного класса, можно с помощью rdf:type. Например,

```
usr:user rdf:type rdfs:Class .
```

Примечание:

Выше мы обозначали тройки непосредственно, заключая их в скобки. Здесь же используется нотация N-Triple, в которой тройка записывается без скобок, но в конце записи ставится точка.

В RDFS также существует класс всех свойств, обозначаемый как rdf:Property. Все свойства являются экземплярами этого класса, а сам он, в свою очередь, является экземпляром класса rdfs:Class. Класс rdf:Property использовался в примере выше.

Для того чтобы сказать, что значения некоторого свойства являются экземплярами некоторого класса, т.е. чтобы задать типы свойств, используется свойство rdfs:range. Выражение:

```
P rdfs:range C
```

означает, что P — это экземпляр класса rdf:Property, а C — экземпляр класса rdfs:Class, и что все ресурсы, входящие в качестве объектов в тройки, предикат которых — это свойство P, являются экземплярами класса C. В ООП эта ситуация соответствует объявлению типа свойства. Например, объявление поля usr:age на языке C++ выглядит следующим образом:

```
int age;
```

задает тип int свойства age, т.е. говорит о том, что все объекты свойства age являются экземплярами класса int. Если rdfs:range позволяет задать тип значений, которые будет принимать некоторое свойство, то свойство rdfs:domain позволяет задать класс, чьим атрибутом является данное свойство. Выражение

```
P rdfs:domain C
```

говорит о том, что P — это экземпляр класса rdf:Property, а C — экземпляр класса rdfs:Class, и что все ресурсы, входящие в качестве субъектов в тройки, предикат которых — это свойство P, являются экземплярами класса C. Если свойство usr:age является атрибутом класса usr:user, то об этом можно сказать так:

```
usr:age rdfs:domain usr:user .
```

Итак, выражение языка C++:

```
namespace usr
{
  class user
  {
    int age;
  };
};
```

можно записать на RDFS следующим образом:

```
usr:age rdfs:range int .
usr:age rdfs:domain usr:user .
```

Свойство rdfs:subClassOf представляет собой аналог наследования в ООП. Выражение

```
C1 rdfs:subClassOf C2
```

означает, что тип C1 является подтипом типа C2, т.е. что каждый экземпляр класса C1 является также и экземпляром класса C2.

В RDFS также определены элементы, позволяющие задавать контейнеры (классы, унаследованные от rdfs:Container) и коллекции (класс rdf:List). Для подробного ознакомления со всеми элементами словаря RDF читатель может обратиться к официальному описанию RDFS на сайте консорциума W3 [15].

Язык OWL

OWL (Web Ontology Language [17]) представляет собой язык, предназначенный для описания онтологий и разработанный консорциумом W3 специально для этих целей. OWL построен как расширение RDF и RDFS. Это означает, что основная конструкция — это тройка языка RDF. В этом контексте язык OWL можно рассматривать как расширенный вариант RDFS, позволяющий не только описывать классы и свойства, но также задавать ограничения на их использование. На языке дескрипционной логики это означает, что логика, лежащая в основе OWL, содержит кроме описания отношений также и аксиомы, задающие соотношения между данными отношениями и различного рода ограничения последних.

Базовым элементом языка OWL является класс всех классов, определяемый как owl:Class. Класс owl:Class —

это экземпляр рассмотренного выше класса rdfs:Class. Любой OWL-класс должен быть задан как экземпляр класса owl:Class. Например, если мы хотим определить класс Human (человек), то должны задать тройку

```
Human rdf:type owl:Class
```

В языке OWL также присутствуют два предопределенных класса:

- Класс owl:Thing (сущность), который обозначает множество всех индивидов.
- Класс owl:Nothing (ничто), обозначающий пустое множество.

Каждый класс OWL является дочерним классом класса owl:Thing и родительским классом класса owl:Nothing. Наследование классов в языке OWL задается с помощью конструкции rdfs:subClassOf, т.е. точно так же, как и в языке RDF Schema.

В OWL существует разделение свойств на два класса:

- Объектные свойства используются для связывания индивидов друг с другом. Объектные свойства — это экземпляры класса owl:ObjectProperty.
- Свойства типов данных связывают индивидов с так называемыми значениями типов данных (data values). Под значениями здесь подразумеваются RDF-литералы, или типы данных, определенные в XML Schema. Свойства типов данных — это экземпляры класса owl:DatatypeProperty.

Классы owl:ObjectProperty и owl:DatatypeProperty являются дочерними классами класса rdf:Property.

Язык OWL позволяет описывать различные характеристики классов и свойств, которые обычно задаются как разного рода ограничения на структуру связей между своими экземплярами. Эти ограничения выражаются в виде предопределенных соотношений, называемых в языке OWL аксиомами. В этом состоит основное отличие языка OWL от RDFS. Эти ограничения позволяют выражать в онтологии более тонкие вещи, чем с помощью RDFS. Например, в языке OWL имеется аксиома owl:equivalentClass, которая позволяет сказать, что множества экземпляров двух данных классов совпадают. Например, можно задать класс с именем ГенеральныйДиректор, а можно еще сказать о классе ГлаваКомпании. Тогда можно определить эквивалентность этих классов следующим образом:

```
ГенеральныйДиректор owl:equivalentClass ГлаваКомпании .
```

Также можно задавать ограничения для атрибутов (свойств) классов. Например, ограничения количества элементов (cardinality constraints) накладывают ограничения на мощность множества значений данного свойства, если в качестве субъекта выступает некоторый конкретный класс. Так, можно указать, что количество значений свойства Игрок класса ФутбольнаяКоманда должно равняться числу 11.

Таких ограничений в языке OWL множество. Но все они подобраны таким образом, чтобы не снизить производительность алгоритма логического вывода по фактам, которые описаны в онтологии. Более подробную информацию о языке OWL читатель может почерпнуть из [16].

Синтаксис XML удобен для чтения машинами, но не слишком удобен для человека. Поэтому обычно документы с онтологиями, написанными на языке OWL, не редактируют непосредственно, а используют для редактирования специализированные программы. Одна из таких программ называется Protege [18]. Эта программа бесплатна для использования и может быть загружена с сайта проекта Protege [18]. OWL-код, записанный в файле онтологии, представляется непосредственно в виде коллекций классов и их свойств, а также в виде различных их ограничений, как это описано выше. Программа Protege предоставляет возможность задавать запросы на языке SPARQL [19] к открытой онтологии, а также производить различные логические манипуляции над онтологией.

Кроме программ редактирования онтологий имеется большое количество реализаций Web-сервисов онтологий — хранилищ библиотек онтологий. Такие хранилища подобны серверам баз данных, предоставляемых разработчиками сервиса программистам, которые наполняют содержимое этих сервисов конкретными данными. Существуют как коммерческие, так и бесплатные реализации таких сервисов. Один из наиболее популярных сервисов такого рода называется OWLIM [20], написан на Java и, как декларируется его разработчиками, представляет собой быстрейшую на сегодняшний день реализацию хранилища. Более подробный список реализаций RDF-хранилищ приведен в [1].

Заключение

В данной работе сделана попытка в популярной форме познакомить читателя с понятием онтологии и приложениями этого понятия в компьютерных системах. Чтобы изложение не было слишком абстрактным, было приведено описание одного конкретного примера использования онтологий в Интернет — описания содержимого Web-страниц в виде онтологий. Для таких описаний консорциумом W3 разработаны специальные языки: RDF и OWL. В статье было приведено их краткое изложение.

К сожалению, ограничения объема, налагаемые на журнальную статью, не дают возможности рассказать о понятии «онтология» подробно. Заинтересованный читатель может получить подробную информацию в книге «Онтологии в компьютерных системах» [1]. В этой книге рассмотрены различные аспекты применения онтологий в информационных системах, и многие из этих применений не ограничиваются Интернет.

Автор надеется, что приведенная в работе информация будет полезна читателю не только в качестве «расширения кругозора», но и даст пищу для реализаций конкретных проектов в самых различных областях информатики.

Список литературы

1. Лапшин В.А. Онтологии в компьютерных системах. М.: Научный мир, 2010.
2. Gruber T.R. The role of common ontology in achieving sharable, reusable knowledge bases // Principles of Knowledge Representation and Reasoning. Proceedings of the Second International Conference. J.A. Allen, R. Fikes, E. Sandewell — eds. Morgan Kaufmann, 1991, 601-602.
3. Бениаминов Е.М. Алгебраические методы в теории баз данных и представлении знаний. М.: Научный мир, 2003.
4. Guarino N. Formal ontology in information systems // Proceedings of FOIS'98, Trento, Italy, 6-8 June 1998. Amsterdam, IOS Press, 1998. 3-15.
5. Когаловский М., Калиниченко Л. Концептуальное моделирование и онтологические модели // Онтологическое моделирование. Труды симпозиума в г. Звенигороде, 19-20 мая, 2008.
6. Описание концепций языка RDF на сайте W3. http://www.w3.org/TR/rdf-concepts/ (http://www.w3.org/TR/rdf-concepts/).
7. The description logic handbook: Theory, implementation, and applications. F. Baader, D. Calvanese, D. McGuiness, D. Nardi, P. Patel-Schneider. Cambridge: University Press, 2003.
8. Quillian M. Word concepts: A theory and simulation of some basic capabilities // Behavioral Science, 1967. 12. 410-430.
9. Minsky M. A framework for representing knowledge. J. Haugeland — ed. Mind Design. The MIT Press, 1981.

10. Uniform resource identifier (URI): Generic Syntax. http://tools.ietf.org/html/rfc3986 (http://tools.ietf.org/html/rfc3986).
11. Uniform resource locators (URL). http://tools.ietf.org/html/rfc1738 (http://tools.ietf.org/html/rfc1738).
12. URN syntax. http://tools.ietf.org/html/rfc2141 (http://tools.ietf.org/html/rfc2141).
13. Спецификация нотации N-Triple на сайте консорциума W3. http://www.w3.org/2001/sw/RDFCore/ntriples/ (http://www.w3.org/2001/sw/RDFCore/ntriples/).
14. RDF Semantisc. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/.
15. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/ (http://www.w3.org/TR/rdf-schema/).
16. Сайт консорциума World Wide Web. http://www.w3.org/ (http://www.w3.org/).
17. Описание языка OWL на сайте W3. http://www.w3.org/TR/owl-ref/ (http://www.w3.org/TR/owl-ref/).
18. Сайт редактора онтологий OWL Protege. http://protege.stanford.edu/ (http://protege.stanford.edu/).
19. SPARQL query results XML Format. http://www.w3.org/TR/rdf-sparql-XMLres/ (http://www.w3.org/TR/rdf-sparql-XMLres/).
20. Сайт OWLIM Semantic Repository. http://www.ontotext.com/owlim/ (http://www.ontotext.com/owlim/).

❖

O lock-free алгоритмах (+бонус)

Начало на стр. 35, 52

```
    cell = &buffer_[pos & buffer_mask_];
    // загружаем статус (sequence) текущего элемента
    size_t seq = cell->sequence_.load(std::memory_order_acquire);
    intptr_t dif = (intptr_t)seq - (intptr_t)pos;
    // элемент готов для записи
    if (dif == 0)
    {
      // пытаемся сдвинуть позицию для добавления
      if (enqueue_pos_.compare_exchange_weak(pos,
            pos + 1, std::memory_order_relaxed)
      )
        break;
      // если не получилось, то начинаем сначала
    }
    // элемент ещё не готов для записи (очередь полна или типа того)
    else if (dif < 0)
      return false;
    // нас кто-то опередил
    // перезагружаем текущий элемент и начинаем сначала
    else /* if (dif > 0) */
      pos = enqueue_pos_.load(std::memory_order_relaxed);
  }

  // в данной точке мы зарезервировали элемент для записи

  // пишем данные
  cell->data_ = data;
  // помечаем элемент как готовый для потребления
  cell->sequence_.store(pos + 1, std::memory_order_release);

  return true;
}

bool dequeue(T& data)
{
  cell_t* cell;
  // загружаем текущую позицию для извлечения из очереди
  size_t pos = dequeue_pos_.load(std::memory_order_relaxed);

  for (;;)
  {
    // находим текущий элемент
    cell = &buffer_[pos & buffer_mask_];
    // загружаем статус (sequence) текущего элемента
    size_t seq = cell->sequence_.load(std::memory_order_acquire);
    intptr_t dif = (intptr_t)seq - (intptr_t)(pos + 1);

    // элемент готов для извлечения
    if (dif == 0)
    {
      // пытаемся сдвинуть позицию для извлечения
      if (dequeue_pos_.compare_exchange_weak(pos, pos + 1,
            std::memory_order_relaxed)
      )
        break;
      // если не получилось, то начинаем сначала
    }
    // элемент ещё не готов для потребления (очередь пуста или типа того)
    else if (dif < 0)
      return false;
    // нас кто-то опередил
    // перезагружаем текущий элемент и начинаем сначала
    else /* if (dif > 0) */
      pos = dequeue_pos_.load(std::memory_order_relaxed);
  }

  // в данной точке мы зарезервировали элемент для чтения

  // читаем данные
  data = cell->data_;
  // помечаем элемент как готовый для следующей записи
  cell->sequence_.store(pos + buffer_mask_ + 1, std::memory_order_release);

  return true;
}

private:
  struct cell_t
  {
    std::atomic<size_t>   sequence_;
    T             data_;
  };

  static size_t const     cacheline_size = 64;
  typedef char        cacheline_pad_t [cacheline_size];

  cacheline_pad_t       pad0_;
  cell_t* const         buffer_;
  size_t const        buffer_mask_;
  cacheline_pad_t       pad1_;
  std::atomic<size_t>     enqueue_pos_;
  cacheline_pad_t       pad2_;
  std::atomic<size_t>     dequeue_pos_;
  cacheline_pad_t       pad3_;

  mpmc_bounded_queue(mpmc_bounded_queue const&);
  void operator = (mpmc_bounded_queue const&);
};
```

И вот реализация подмножества C++0x std::atomic, необходимого для компиляции очереди (MSVC, x86-32):

```
#include <intrin.h>

enum memory_order
{
  memory_order_relaxed,
  memory_order_consume,
  memory_order_acquire,
  memory_order_release,
```

Окончание на стр. 72