Что такое RDF и для чего он хорош?

Эта статья является введением в RDF (Resource Description Framework — модель описания ресурсов), стандарт для кодирования метаданных и других видов знаний в семантическом вебе (Semantic Web). Благодаря семантическому вебу компьютерные программы могут использовать возрастающие объёмы структурированных данных, распределённо и децентрализованно рассеянные по сети в настоящее время. RDF представляет собой абстрактную модель, обеспечивающую способ разбиения знаний на дискретные части. Хотя наиболее известным синтаксисом является RDF/XML, RDF можно хранить в разных форматах. В этой статье рассматриваются: абстрактная модель RDF, два конкретных формата для его представления в последовательной форме, способы использования RDF и его отличия от чистого XML, высокоуровневая семантика RDF, лучшие методы устройства источников RDF-данных и выполнения запросов к ним.

Данный документ был изначально написан в октябре 2005 года. В июле 2006 года он был пересмотрен и расширен материалом из опубликованной мной на xml.com статьи 'Что такое RDF' ('What is RDF'). В январе 2008 года была обновлена информация о N3 и RDF/XML и появились новые разделы о связанных данных (Linked Data) в веб и о запросах к базам данных семантического веба.

Почему мы нуждаемся в новых стандартах для семантического веба?

Используя семантический веб компьютеры самостоятельно собирают для нас данные. Семантический веб ('SemWeb') позволяет им разыскивать распределенные в сети знания, связывать их вместе, и затем использовать по назначению. Говоря на языке аналогий, веб в его нынешнем проявлении — это децентрализованная платформа по распространению визуальных представлений, в то время как семантический веб — децентрализованная платформа для распределённых знаний. RDF является стандартом W3C для кодирования знаний.

Конечно, знания присутствуют в вебе и сейчас, но они не предназначены для компьютеров. Рассмотрим любую страницу в Википедии. Она может содержать исчерпывающую информацию для просматривающего её человека. Но для компьютера, (отображающего страницу) предназначена лишь разметка для воспроизведения страницы. Степень, в которой HTML, изображения, Flash и т.п. имеют смысл для компьютера, почти всегда определяется одной целью — обеспечение воспроизведения для конечного пользователя. Настоящее содержимое файлов — знания — доступны только человеку и недоступны компьютеру.

'Семантика' в названии 'семантический веб' не обозначает, что компьютеры в буквальном смысле этого слова начнут понимать смысл. Подразумевается, что компьютеры будут механически манипулировать логически разделимыми смысловыми фрагментами для получения полезного результата.

Теперь представим себе новый Интернет, в котором настоящим содержимым могут манипулировать компьютеры. Проиллюстрируем это как сеть баз данных. На одном 'семантическом' сайте публикуется база данных о линейке продуктов (продукты и их описания). В то же время на другом сайте публикуется база данных, содержащая обзоры продуктов. На третьем сайте для розничной торговли публикуется база данных продуктов на складе. Какие стандарты могут упростить написание приложений для совместного использования распределённых баз данных, обеспечив возможность компьютерной обработки данных из этих трёх источников таким образом, чтобы помочь конечному пользователю принять более обоснованное решение о покупке?

Как когда-то отсутствие XML не могло остановить обмен информацией, так и сейчас ничто не может помешать написанию приложений для решения подобных задач. Но стандарты призваны облегчить разработку программ (особенно программ, функционирующих в децентрализованных

системах). Существуют некоторые утверждения, касающиеся распределённых знаний, которые, на наш взгляд, стали общепризнанными.

1. Файлы в семантическом вебе должны гибко предоставлять информацию. Жизнь невозможно упаковать в таблицу, как в реляционных базах, или в иерархическое древо, как в XML. Информацию о кинокартинах или развлекательных передачах на телевидении, представленную в виде графа на рисунке ниже, на самом деле лучше всего представлять именно в виде графа.

[[рисунок: Знания в форме графа.]]

Мы, конечно же, не можем рисовать наш 'путь' через семантический веб. Вместо этого нам нужен табличный способ записи этих графов. Сравните расположенную ниже таблицу с рисунком выше. Каждая строка представляет собой стрелку (дугу графа) на рисунке. В первой колонке содержится имя узла, из которого исходит дуга. Вторая колонка хранит в себе метку ребра (его тип). В третьей колонке — имя узла, на который указывает стрелка.

[[таблица: Начальный узел, метка дуги и конечный узел.]]

Оба способа записи представляют то, что с абстрактной точки зрения называется графом. Представляя граф в виде изображения или таблицы, мы говорим об одном и том же. Подробнее об этом ниже.

2. Файлы в семантическом вебе должны соотноситься друг с другом. Прайс-лист, размещенный продавцом, и независимо от него опубликованный потребителем обзор товаров, должны включать в себя способ определения того, что речь идет об одном и том же продукте. Использование только имени продукта недостаточно. В мире может существовать два продукта с именем 'Супер Штука 3000', а мы хотим избежать неоднозначности в семантическом вебе, чтобы компьютеры могли обрабатывать информацию с уверенностью. Для семантического веба нужны глобальные уникальные идентификаторы, которые могут быть назначены децентрализовано.

3. Мы будем использовать словари терминов для того, чтобы делать утверждения о сущностях. Эти словари должны допускать возможность совместного использования, когда речь идет об одной и той же сущности. Например, если говорить об актере, который снимался как в телешоу, так и в кинофильмах, должна быть возможность совместного использования в рамках одного и того же документа лексики телешоу, которая в ходу у их приверженцев, и, независимо разработанной терминологии, принятой в среде киноманов.

Как мы увидим в следующем разделе, RDF (модель описания ресурсов) позволяет стандартизировать эти требования. Перед тем, как углубиться в слишком абстрактные вещи, рассмотрим RDF на примерах, воспользовавшись представленным чуть выше графом. Сначала остановимся на формате Нотации 3 (N3), который наиболее близок к табличному представлению графа:

[[пример: Пример нотации 3.]]

Также рассмотрим стандартный формат RDF/XML, который, возможно, легче интуитивно почувствовать, и который более выразительно отражает иерархическую структуру графа. Но в большинстве случаев он имеет тенденцию усложнять лежащий в основе граф:

[[пример: Пример RDF/XML.]]

RDF изначально был создан в 1999 году в качестве основанного на XML стандарта описания метаданных. Буквально, метаданные — 'данные о данных'. Метаданными называют, например, сведения об авторе веб-страницы, дату очередной публикации в блоге, и тому подобную информацию, которую можно считать вторичной по отношению к основному содержанию. С момента своего рождения (и особенно после выхода обновленной спецификации 2004 года) RDF

расширял рамки своего назначения. Наибольшим обобщением является применение RDF не только для представления информации о ресурсах в сети, но также для описания объектов реального мира и их взаимосвязей: людей, мест, понятий и т.д.

Введение в RDF

Если вы не знаете достаточно хорошо, что такое RDF, то для вас, возможно, будет лучше постараться забыть то, что вам уже известно о RDF, на то время, пока вы будете читать оставшуюся часть этого раздела. RDF существует как пересечение нескольких различных технологий, поэтому легко прийти к выводу, что RDF это всего лишь особый формат XML или инструмент для наполнения блогов. Забудьте то, что вы знаете. Здесь рассказывается о RDF, начиная с самых основ.

RDF — это универсальный метод разделения знания на маленькие части, в соответствии с некоторыми правилами, учитывающими семантику (смысл) этих частей. Суть в том, что такой метод должен быть достаточно простым, чтобы с его помощью можно было описать любой факт, и достаточно структурированным, чтобы представить факт в такой форме, в которой компьютерные приложения смогут осуществлять полезные действия со знаниями, выраженными в формате RDF. Я обозначил RDF как 'метод', что в определённом смысле точнее, чем 'формат', поскольку упомянутые части можно записать многими способами, сохранив при этом изначальную информацию и структуру. Точно так же, как некоторое значение можно выразить на разных естественных языках или реализовать некоторые структуры данных разными способами.

Между RDF и XML можно провести некоторые аналогии. В основу XML тоже заложены принцип простоты и поддержка данных любого типа. XML тоже представляет собой нечто большее, чем просто формат файла. XML — это основа для манипулирования иерархическими, самоописываемыми документами, как сохранёнными на диске в обычном формате с угловыми скобочками и знаками слеш (называемом 'brackets-and-slashes'), так и хранимыми в памяти.

Отличает RDF от XML то, что RDF разработан для представления знаний в распределённом мире. Тот факт, что RDF разработан для знаний, а не для данных, означает, что RDF отчасти имеет отношение к смыслу. Всё, что упомянуто средствами RDF, содержит некоторое значение. Это может быть ссылка на что-либо в реальном мире, например, человека или действие, или это может быть абстрактное понятие, например, отношение дружбы. Стандарт RDF описывает, как с помощью объединения трёх таких сущностей можно констатировать некоторый факт. Значение триплета '(Джон, Боб, отношение дружбы)' заключается в том, что Джон и Боб являются друзьями. Совокупность фактов в достаточном количестве может являться некоторой формой представления знаний. Стандарты, основанные на RDF, включая RDFS и OWL, дополняют семантику RDF, делая возможным построение логических выводов на основе совокупности данных.

Продолжая сравнение, отметим, что сам XML имеет не очень большое отношение к смыслу, значению. XML-узлы не обязаны быть связанными с какими-либо обособленными понятиями, а сам XML стандарт не указывает на то, каким образом извлекать факты из документа. К примеру, если вы ознакомились с некоторым количеством XML-документов, в которых корневые узлы названы на неизвестном вам иностранном языке, вы не сможете сделать ничего полезного с такими документами, кроме как отобразить их. RDF-документы с узлами, смысла которых которые вы не можете понять, тем не менее, могут быть действительно успешно обработаны, потому что RDF определяет некоторый начальный уровень смысла. Вышесказанное не значит, что вы не можете разработать свой собственный стандарт на основе XML, в котором будет оговорен порядок извлечения набора фактов из XML-документа. Но, осуществив это, вы, возможно, обнаружите, что просто изобрели заново что-то вроде RDF.

Второй ключевой аспект RDF в том, что эта технология хорошо работает с распределённой информацией. Это значит, что RDF-приложения могут 'собрать' вместе RDF-файлы, опубликованные разными людьми по всему Интернету, и легко получить из них новые факты, которые не утверждаются ни в одном отдельно взятом документе. Достигнуть этого можно двумя

путями. Во-первых, связывая документы вместе, благодаря использованию в них общих словарей. Во-вторых, позволяя в любом документе использовать любой словарь. Это даст невероятную гибкость для представления фактов о широком диапазоне вещей, черпая информацию из широкого круга источников.

[[блок: Для ознакомления с официальной документацией по RDF начните с 'RDF Primer'.]]

Знания в виде триплетов

RDF обеспечивает универсальный, гибкий метод декомпозиции знаний на маленькие части, называемые триплетами, по некоторым правилам, учитывающим семантику (смысл) этих частей.

Основа метода в разбиении знаний на части и составлении того, что принято называть помеченным направленным графом.

Каждая дуга такого графа представляет собой некоторый факт, отношение между двумя сущностями. На уже упоминавшемся рисунке дуга с меткой starred_in (в ролях), соединяющая узлы vincent_donofrio и the_thirteenth_floor представляет собой утверждение о том, что Винсент Д'Онофрио (Vincent D'Onofrio) снимался (в ролях) в фильме 'Тринадцатый этаж' ('The Thirteenth Floor'). Факт (или утверждение), представленный таким способом, состоит из трёх частей: субъект, предикат (аналогичен глаголу в естественных языках) и объект. Субъект представляется узлом, из которого исходит дуга. Значение предиката определяется типом дуги (обозначается с помощью метки дуги). Объект — это узел, к которому направлена дуга.

Из шести документов, составляющих спецификацию RDF, можно узнать о двух вещах. Во-первых, об основных принципах абстрактной модели, то есть о том, как использовать триплеты для представления знаний о мире. Во-вторых, о том, как представить эти триплеты в виде XML. Далее мы рассмотрим обе эти темы.

Абстрактная модель RDF: высказывания

RDF — это не что иное, как универсальный метод разбиения информации на части. В этом определении подчёркивается тот факт, что один и тот же метод может быть использован для любого типа информации. А метод состоит в следующем: представить информацию как список утверждений вида СУБЪЕКТ ПРЕДИКАТ ОБЪЕКТ. Субъект и объект указывают на две сущности в реальном мире, а предикат определяет отношение между ними. Вы можете представлять себе предикаты как глаголы.

Вот как я представил бы информацию о моей квартире в виде RDF-высказываний:

СУБЪЕКТ ПРЕДИКАТ ОБЪЕКТ

Я владею моей_квартирой

моя_квартира содержит мой_компьютер

моя_квартира содержит мою_кровать

моя_квартира находится_в Филадельфии

Эти четыре строки выражают четыре факта. Каждая строка называется 'высказывание' или 'триплет'.

В качестве субъектов, предикатов и объектов в RDF выступают имена для сущностей: для конкретных сущностей (например, 'моя_квартира'), или для абстрактных (например, 'содержит'). Эти имена не имеют внутренней структуры, и сами по себе не имеют смысла. Они используются

как имена собственные или как переменные. Не важно какое именно имя вы выберете для той или иной сущности, главное везде использовать его согласованно.

Имена в утверждениях RDF обозначают сущности реального мира, тем самым позволяя на них ссылаться. То, что обозначают эти имена, может называться ресурсами (из-за применения RDF для описания метаданных веб-ресурсов), узлами (как в теории графов), или сущностями. Все эти термины являются синонимами. Например, имя 'моя_квартира' обозначает мою настоящую квартиру, которая является сущностью в реальном мире. Есть небольшое, но важное различие между именами и сущностями, которые они обозначают. Например, два разных имени могут быть использованы в качестве ссылок на одну и ту же сущность.

Любой предикат — это отношение между двумя сущностями. 'Владею' — это отношение между владельцем и предметом владения. 'Содержит' — отношение между контейнером и тем, что контейнер содержит. 'Находится_в' — обратное отношение, между тем, что содержится в контейнере, и контейнером. Порядок расположения субъекта и объекта в RDF очень важен.

Следующий аспект RDF почти очевиден, но я хочу отметить его: если кто-то ссылается на что-то как на X в одном месте, и при этом X используется в другом месте, то оба X обозначают одну и ту же сущность. Если я написал 'моя_квартира' в первой строке, то используя это обозначение в последующих трех строчках, я подразумеваю всё ту же самую квартиру.

Правила, которые описаны выше, уже позволяют намного больше, чем вы можете себе представить. Взяв эту таблицу утверждений, я могу написать простую программу, которая сможет отвечать, например, на такие вопросы как: 'кто владеет моей_квартирой?' или 'что содержит моя_квартира?'. Каждый такой вопрос соответствует форме RDF-утверждения, за исключением того, что вопросительные слова (такие как 'кто' и 'что') должны интерпретироваться программой в качестве символов-шаблонов. Простая программа, отвечающая на вопросы, может просто сравнивать вопрос с каждым из утверждений в таблице. Каждое совпадение является ответом. Вот псевдокод:

[[псевдокод: Псевдокод программы отвечающей на вопросы.]]

Для того, чтобы принести пользу, компьютеру нет необходимости знать, что на самом деле означают слова в английском языке. Поэтому, автор программы волен выбирать подходящие имена для сущностей (например: моя_квартира) и использовать правильные предикаты (владеть, содержать) по своему усмотрению. Инструменты RDF ничего не знают о том, какой смысл скрывается за этими именами, тем не менее, они могут с пользой обрабатывать информацию. (Ниже я дам больше информации о полезных возможностях.)

URI в качестве имён ресурсов

Информация в форме RDF предназначена для публикации в Интернете, поэтому использованные мной выше имена несут в себе проблему. Я не могу именовать что-либо 'моя_квартира', потому что кто-нибудь другой также может использовать имя 'моя_квартира' для своей квартиры. Следуя рассмотренному выше правилу RDF, RDF-инструменты интерпретировали бы обе сущности 'моя_квартира' как ссылающиеся на один и тот же объект в реальном мире, тогда как на самом деле подразумеваются две различные квартиры. Последний аспект RDF заключается в том, что имена должны быть глобальными, для уверенности, что вы не выберите имя, которое кто-то другой может также использовать для обозначения чего-либо другого. Формально имена для субъектов, предикатов, и объектов должны являться единообразными идентификаторами ресурсов (Uniform Resource Identifier, сокращённо URI).

На сегодняшний день в мире семантического веба URI используются не вполне последовательно, так что будьте снисходительны ко мне.

С одной стороны, предполагается, что URI непрозрачны. URI могут иметь такой же синтаксис или

формат как и адреса веб-сайтов, поэтому вы увидете RDF-файлы, содержащие такие URI как http://www.w3.org/1999/02/22-rdf-syntax-ns#type в качестве глобального имени какой-нибудь сущности. Но тот факт, что они выглядят как веб-адреса, совершенно случаен. И не важно, существует на самом деле или нет сайт по этому адресу. Кроме начинающихся с 'http:' существуют другие типы URI. URN — это тип URI, который используется для таких вещей, как идентификация книг по их номеру ISBN, например, urn:isbn:0143034650. TAG — это URI общего назначения. Они выглядят следующим образом: tag:govtrack.us,2005:congress/senators/frist. В этой статье будут использованы все три типа URI.

Вне зависимости от их формы, URI, которые вы видите в RDF-документах — это просто состоящие из слов названия для сущностей и ничего более.

Это был один взгляд на URI. С другой стороны, в последние годы (начиная с 2007) обычно ожидается, что если вы создаёте начинающийся на 'http:' или другой разыменовываемый URI, то на самом деле размещаете что-либо по этому адресу, чтобы RDF-клиенты могли обратиться к странице и получить с неё некоторую информацию. То есть: то, что URI означает в документе не зависит от того, чем этот URI является. Но если вы используете URI, который можно разыменовать, то другие могут ожидать, что вы разместите что-то по указанному адресу в сети. Мы ещё вернёмся к этому в разделе о связанных данных.

URI используются как глобальные имена потому, что они позволяют разбить пространство всех возможных имён на элементы, имеющие явных владельцев. URI, начинающиеся с http://govtrack.us/ очевидно контролируются мной, или кем-либо, кто управляет сайтом по этому адресу. По традиции, если существует очевидный владелец URI, никто кроме владельца не будет создавать новые ресурсы с этим URI. Это предотвращает конфликты имён. Если вы создаёте URI в пространстве имён, которое контролируете, то можете быть уверены, что больше никто не будет использовать тот же URI для обозначения чего-то другого. (Конечно, кто-нибудь может использовать ваши URI не так, как вы задумали, но это тема для другой статьи.)

Поскольку URI могут быть достаточно длинными, то в различных RDF-нотациях они обычно сокращаются посредством использования концепции пространств имён, как в XML. Как и в XML, пространства имён обычно задаются в начале RDF-документа и используются для сокращений в этом документе. К примеру, я задал сокращение 'taubz' для URI http://razor.occams.info/index.html#. Во многих RDF-нотациях я теперь могу сократить такие URI как http://razor.occams.info/index.html#моя_квартира, заменяя URI пространства имён (точно в таком виде, как оно было задано) его сокращением и двоеточием, в результате чего получится taubz:моя_квартира. Точные правила для пространств имён зависят от используемого синтаксиса записи RDF.

Важно, что пространства имен не занимают никакого важного положения в RDF. Они просто являются инструментом для более короткой записи длинных URI.

Я могу переписать таблицу о моей квартире так, как показано ниже, заменив простые имена, которые я использовал ранее, на произвольные URI:

[[пример: RDF о 'моей квартире'.]]

Предложенная выше таблица является представленным в вольной форме в табличном виде информационным графом, существующем на абстрактном уровне. Этот же граф представлен в графической форме на рисунке ниже. В дальнейшем мы более подробно рассмотрим стандартные способы записи RDF.

[[рисунок: RDF в виде графа.]]

Подытожим вышесказанное

Вот и весь RDF. Все остальное в семантическом вебе построено на рассмотренных трёх правилах. Повторим их здесь, чтобы довести до сознания всю простоту системы:

1. Факт представляется в форме триплета (субъект, предикат, объект).

2. Субъекты, предикаты и объекты задаются именами конкретных или абстрактных сущностей реального мира.

3. В роли имён используются URI, являющиеся непрозрачными и глобальными.

Эти концепции формируют абстрактную модель данных RDF для кодирования знаний. Такая модель является аналогом типового API, который предоставляют большинство XML-библиотек. Если бы мы не были такими любопытными людьми и не совали бы все время свой нос в файлы, настоящий формат XML не имел бы для нас никакого значения до тех пор, пока у нас были бы наши методы appendChild, setAttribute, и т.д. Конечно, нам нужен общеупотребительный файловый формат для обмена данными, и фактически существует два таких формата для RDF, которые мы рассмотрим далее.

Анонимные узлы и литеральные значения.

На самом деле, конечно, RDF не исчерпывается тремя приведенными выше правилами. Ранее я описал три типа понятий в RDF: ресурсы (сущности или концепты), существующие в реальном мире; глобальные имена для ресурсов (то есть URI); и RDF-высказывания (триплеты или строки в таблице). Существуют еще два понятия.

Литералы

Первым рассмотрим понятие 'литеральное значение'. Литеральные значения представляют собой просто текст, который может быть использован вместо объектов в RDF-триплетах. В отличии от имен (то есть URI), которые заменяют собой сущности в реальном мире, литеральные значения — это просто текстовые данные вставленные в граф. Литеральные значения могут быть использованы для того, чтобы установить связь между людьми и их именами, книгами и их номерами ISBN, и т.д.

[[пример: Некоторые способы использования литеральных значений]]

Безымянные/анонимные узлы

Кроме того, существуют анонимные или безымянные узлы (blank nodes или bnodes). Эти термины являются синонимами. Слова 'анонимный' или 'безымянный' указывают на то, что эти узлы в графе не имеют собственного имени, в силу того, что автор документа не знает имя, не хочет или не может его предоставить. В определенном смысле, это подобно таким высказываниям как: 'Джон дружит с кем-то, но я не скажу с кем'. Когда мы говорим, что эти узлы не имеют имени, нужно иметь в виду две вещи. Во-первых, сущность реального мира, на которую ссылается данный узел, в свою очередь, не обязана быть безымянной. Друг Джона из нашего примера все-таки имеет имя. Во-вторых, когда мы говорим здесь 'безымянный', мы подразумеваем концепцию именования сущностей с помощью URI. На самом деле, анонимные узлы в документе могут иметь 'локальные' идентификаторы, так что на них можно ссылаться несколько раз внутри документа. Безымянность лишь означает, что эти локальные идентификаторы, очевидно, не являются глобальными, и не имеют смысла вне того документа, в котором они использованы.

Вот пример использования литеральных значений и анонимных узлов. 'Джошуа Тоберер' ('Joshua Tauberer') в примере — это литеральное значение, а _:anon123 — это анонимный или безымянный узел.

[[пример: Анонимные узлы и литеральные значения.]]

Чтобы различать URI, имена, привязанные к пространствам имен (сокращенные URI), анонимные узлы и литеральные значения, я использую следующее общепринятое соглашение:

* Полные URI заключаются в угловые скобки.

* Имена, привязанные к пространствам имен, записываются непосредственно, но название пространства имен отделяется двоеточием.

* Анонимные узлы записываются так же, как и имена, привязанные к пространствам имен, но в качестве пространства имён используется зарезервированное значение '_', соответствующее локальное имя записывается после двоеточия.

* Литеральные значения заключаются в кавычки.

Вам потребуется некоторое время, чтобы попробовать изобразить граф, описанный в таблице. Нарисуйте стрелки между узлами.

В этом примере один анонимный узел _:anon123. Мы знаем об этом ресурсе, что он автор ресурса <urn:isbn:0143034650>, и то, что его имя Лоуренс Лессиг (Lawrence Lessig). Так как для этого ресурса не используется никакого глобального имени, мы не можем быть уверенными относительно того, о ком на самом деле мы здесь говорим. И если мы захотим сказать что-то еще о чём-либо, связанном с ресурсом _:anon123, мы должны будем сделать это здесь же, в этом RDF документе, потому что у нас нет возможности сослаться на этого самого Лоуренса Лессига за пределами нашего документа.

Подробнее о литералах: дескрипторы языков и типы данных

Литеральные значения могут дополняться одним из двух возможных элементов метаданных. Первый это дескриптор языка, который указывает, на каком языке представлено текстовое значение литерала. Дескриптор языка нужно рассматривать как рудимент, оставшийся с тех пор, когда RDF только начал использоваться. Сегодня он выглядит как уродливая поделка. Вы можете встретить 'chat'@en литеральное значение 'chat' с английским дескриптором языка, или 'chat'@fr то же самое значение с французским дескриптором языка.

Кроме того, литеральные значения могут быть помечены с помощью URI, идентифицирующего тип данных. Тип данных указывает, как интерпретировать текст литерала: как число, URI, дату или время, и т.д. Типом данных может быть любой URI, но обычно для этого используются типы данных, определенные в XML Schema. Для обозначения типа данных обычно используется запись следующего вида: после литерального значения в кавычках следуют два символа 'крышка' ('^') и URI, определяющий тип данных (может быть указан в сокращённой форме).

[[пример: Типы данных.]]

Понятие типа данных не такое простое, как может показаться на первый взгляд. Давайте рассмотрим тип данных для чисел с плавающей точкой. Сами числа с плавающей точкой (как абстрактное математическое понятие) отличаются от текста, который мы используем для их представления на бумаге. Например, текст '5.1' представляет число 5.1, которое также можно обозначить как '5.1000' и '5.10'. Пример показывает, что существует множество текстовых представлений (так называемых лексических представлений) для одного и того же значения. Тип данных говорит нам как преобразовывать лексическое представление в значение и наоборот.

Семантика RDF охватывает дескрипторы языков и типы данных. Это обозначает две вещи. Во-первых, литеральное значение без дескриптора языка и без типа данных отличается от литерала с дескриптором языка и отличается от литерала с типом данных. Эти четыре высказывания являются разными утверждениями, и ни одно из них не является тождественным какому либо другому:

[[пример: Семантика литералов.]]

Иначе говоря, нетипизированный литерал с дескриптором языка или без него — это не то же самое, что типизированный литерал. Второй момент, касающийся семантики литералов, заключается в том, что два типизированных литерала, которые выглядят различными, могут быть равны друг другу, если их типы данных отображают их лексическое представление в одно и то же значение. Следующие два утверждения эквивалентны (по крайней мере для RDF приложений, поддерживающих семантику типов данных XSD):

[[пример: Семантика типов данных.]]

Это значит, что Джону 10 лет. Пример демонстрирует, что текстовое представление числа не имеет отношения к делу и не является частью закодированного триплетом информационного содержания. Заметим, что если бы тип данных чисел с плавающей точкой не был указан, то триплеты не были бы эквивалентными, и текстовое представление числа 10 стало бы частью информационного содержимого.

Подробнее об анонимных узлах: некоторые предостережения

В отличие от глобальных URI, локальные идентификаторы, используемые для обозначения анонимных узлов, явным образом выражают не-глобальное имя. Одинаковые локальные идентификаторы анонимных узлов, используемые в двух разных документах, могут относиться к двум разным сущностям. Кроме того, идентификатор сам по себе случаен, и конкретный идентификатор, используемый в любом частном случае, не является частью информационного содержания самого документа.

В зависимости от контекста анонимные узлы могут трактоваться с небольшим отличием. В запросах их можно рассматривать как переменные. Иногда они могут использоваться для описания ограничений (но это отдельная тема).

Анонимные узлы часто используют для того, чтобы избежать указания URI людей, как продемонстрировано в примере выше. Они также часто используются для представления более сложных отношений:

[[пример: Анонимные узлы для сложных отношений.]]

В этом примере анонимный узел использован в качестве промежуточного шага при описании отношения между мной и частями моего имени. Узел позволяет представить моё имя в структурированном виде, что более точно, чем одно непрозрачное литеральное значение 'Joshua Ian Tauberer'. RDF допускает только бинарные отношения, поэтому для выражения множественных отношений необходимо использовать промежуточные узлы, в качестве которых часто применяют анонимные.

Чтение и запись RDF

В этом разделе мы рассмотрим два стандартных способа записи RDF. Поскольку RDF является абстрактной моделью, основанной на графах, то рассматриваемые форматы называются синтаксисом записи RDF в последовательной форме.

Нотация 3

Нотация 3 ('N3') или подмножество, называемое Turtle, фактически является стандартом записи RDF. Этот синтаксис не является стандартом W3C, но широко распространён, обычно используется при обсуждениях в списках рассылки разработчиков семантического веба. Этот синтаксис важен для понимания, поскольку наиболее понятным образом охватывает абстрактный граф.

Вот пример документа, позволяющий легко понять, какие триплеты закодированы.

[[пример: Пример нотации 3.]]

В табличной форме (обычно называемой NTriples) это представляется так:

[[пример: Табличная форма или NTriples.]]

В этом примере представлены утверждения о географических координатах (широте и долготе) Принстонского университета. А также о том, что в университете есть факультет, который называется 'Факультет информатики'.

В N3 и Turtle утверждения записываются в следующем порядке: сначала URI субъекта (записывается в скобках, либо предваряется названием пространства имён), потом следует URI предиката, затем URI объекта или литеральное значение, в конце ставится точка. Все пространства имён объявляются в самом начале и предваряются директивой @prefix. В такой сокращённой записи полные URI могут быть собраны вновь, например для geo:lat соединение (конкатенация) соответствующего названия пространства имён со второй частью сокращённой записи выглядит так: http://www.w3.org/2003/01/geo/wgs84_pos# + lat = http://www.w3.org/2003/01/geo/wgs84_pos#lat.

N3 обладает некоторыми приятными свойствами, позволяющими ещё сильнее сокращать запись. Если во многих утверждениях повторяются субъект и предикат, то можно просто отделять объекты друг от друга запятыми:

[[пример: Приятные особенности синтаксиса N3: запятые.]]

Если один и тот же субъект повторяется, но с различными предикатами, можно использовать точку с запятой, как в примере:

[[пример: Приятные особенности синтаксиса N3: точка с запятой.]]

В N3 есть ещё некоторые возможности сокращения записи. Обычный предикат rdf:type может быть сокращён и представлен в виде буквы 'a' (что можно запомнить как сокращение от 'is a').

Кроме того, анонимные узлы представимы в N3 двумя способами. В соответствии с первым способом, им даются имена в локальном пространстве имён, обозначаемом с помощью префикса '_':

[[пример: Анонимные узлы в нотации 3 — первый метод.]]

Второй способ использует квадратные скобки и является рекурсивным. Запись в квадратных скобках представляет собой анонимный узел, и внутри скобок вы можете разместить пары вида предикат-объект, разделённые знаками точка с запятой, таким образом закодировав свойства анонимного узла. Например:

[[пример: Анонимные узлы в нотации 3 — второй метод.]]

Оба примера с анонимными узлами содержат в точности одну и ту же информацию. Имейте в виду, что форма синтаксиса N3 документа не имеет какого либо значения. При использовании запятых, точек с запятой, скобок, знаков подчёркивания всегда в конце ставится точка. Это имеет значение при чтении закодированных в документе триплетов.

[[блок: Вы можете почитать 'Primer' для ознакомления с другими примерами использования синтаксиса N3.]]

RDF/XML

Спецификации W3C определяют формат применения XML для кодирования RDF. Поскольку он ориентирован на ту же абстрактную модель, что и нотация 3, различия между форматами незначительны и заключаются в читаемости.

Та же информация, что в первом примере N3, в RDF/XML выглядит так:

[[пример: Пример RDF/XML.]]

Сразу возникают вопросы: каковы стандартизованные правила для интерпретации узлов и атрибутов в этом документе в качестве абстрактного RDF-графа и каковы стандартизованные правила для записи документа в этой форме при наличии абстрактного RDF-графа?

В RDF/XML-документе два типа XML-узлов: 1) узлы ресурсов и 2) узлы свойств. Узлы ресурсов являются субъектами и объектами RDF-утверждений. Обычно в XML узлами ресурсов являются теги rdf:Description, имеющие атрибут rdf:about, указывающий на ресурс, который они представляют. В этом примере узлы rdf:Description — узлы ресурсов.

Узлы ресурсов содержат внутри себя узлы свойств (и ничего другого). Каждый XML-узел свойства представляет единственное утверждение. Субъектом утверждения является внешний XML-узел ресурса, содержащий данное свойство. В этом примере четыре утверждения. Субъект первых трёх из них — http://princeton.edu, а субъект четвёртого — http://www.cs.princeton.edu. URI предикатов в четырёх утверждениях (сокращённо): geo:lat, geo:long, edu:hasDept и dc:title.

Как упоминалось ранее, утверждения могут включать в себя в качестве объектов ресурсы или литералы. Чтобы включить литерал в качестве объекта в узел свойства, его значение просто записывается внутри XML-элемента. Выше так использованы литералы '40,35' и '−74,66'.

Использовать в качестве объекта ресурс можно двумя способами. Первый способ представлен в примере — использование атрибута rdf:resource, в который записывается URI объекта. Вы можете описать свойства этого объекта в другом месте, как это сделано выше, в отдельном узле ресурса rdf:Description на верхнем уровне вложенности документа.

Второй способ представляет собой размещение элемента rdf:Description прямо внутри XML-узла свойства и отбрасывание rdf:resource. Пример выше может быть равнозначно записан следующим образом:

[[пример: Альтернативный пример RDF/XML.]]

Мы рассмотрели, как в соответствии со спецификацией взять XML-документ и получить из него таблицу утверждений. Вы можете проверить, понимаете ли формат, сравнив этот документ с первым примером N3 и объяснением из предыдущего раздела. В обоих документах записаны одинаковые триплеты.

Здесь мы рассмотрим только одно синтаксическое сокращение, поскольку оно довольно часто используется. Посмотрите на пример:

[[пример: Пример RDF/XML с указанием типов.]]

В примере содержится только один триплет. Предикат rdf:type используется, чтобы показать к какому типу относится ресурс. В данном примере смысл утверждения в том, что ресурс, обозначенный http://www.princeton.edu, является университетом. Поскольку rdf:type используется часто, то, вместо создания XML-узла свойства для него, можно использовать сокращение, заменив тег rdf:Description на тип:

[[пример: Пример RDF/XML с указанием типов с применением сокращений.]]

Оба документа абсолютно равнозначны. Поэтому, если вместо rdf:Description именем XML-узла ресурса является что-то другое, значит так записан триплет с предикатом rdf:type. rdf:Description сам по себе типом не является. Это часть синтаксиса. Она не переводится в триплет.

В RDF/XML также можно использовать безымянные узлы. Существует много синтаксических сокращений, которые, как вы увидите, широко используются. Чтобы узнать об этом подробнее, ознакомьтесь со спецификацией. А пока продолжим.

Используйте валидатор!

RDF состоит из триплетов. Когда приложение использует RDF в формате XML или N3, оно видит триплеты. Иерархическая структура XML и порядок его узлов теряются при переходе к таблице триплетов. Эта информация, также как пробелы и отступы, не является частью информационного содержимого, закодированного в RDF. Все нюансы синтаксиса XML и N3, а также выбор форматирования не существенны. Убедитесь, что понимаете триплеты, закодированные в любом документе!

Когда есть сомнения, вы можете использовать онлайн-валидатор для преобразования RDF из одного формата в другой, более ориентированный на триплеты. Преобразование RDF/XML в N3, или, ещё лучше, в N-Triples — отличный способ чётко понять, что за триплеты записаны в XML.

W3C предоставляет RDF-валидатор, превращающий в таблицу триплетов RDF-документ, вставленный вами в текстовое поле. Это хорошее средство также и для проверки синтаксиса вашего RDF.

Также на сайте W3C есть валидатор, способный преобразовывать N3 в RDF/XML и наоборот. Вы можете его использовать, чтобы проверить синтаксис RDF/XML или N3, а затем проверить, что за триплеты записаны в N3-документах.

[[блок: Смотри также: официальные определения синтаксиса N3 и RDF/XML.]]

Распределённая информация

Сначала RDF может показаться неоправданно сложным. Для всех примеров, приведённых выше, информация точно так же могла бы быть записана на чистом XML. Если брать один отдельный документ — это действительно так. Но сила RDF проявляется, когда каждый документ рассматривается как часть большой картины, огромного графа знаний, раскиданного по интернету. В этом 'семантическом вебе' каждое RDF-утверждение, размещённое где-либо в сети, является маленьким кусочком большого знания.

Но не каждый кусочек знания в семантическом вебе может быть автоматически понят. Приложениям нужно объяснить, что делать с конкретными предикатами, иначе RDF будет просто кашей из URI. В следующем разделе я рассмотрю этот вопрос подробнее.

Выбор правильных предикатов

Как вы уже видите, RDF может быть использован для моделирования любого типа знаний без необходимости использовать понятия, утверждённые централизованно. Если никто не создал URI для чего-то, что вы хотите описать, то вы можете создать для этого собственный URI. Это относится не только к субъектам и объектам, но также и к предикатам. В вышеприведённом примере я выдумал все URI, и это, тем не менее, абсолютно правильный RDF.

Проблема в том, что если я выдумаю все URI, мой RDF-документ не будет иметь смысла ни для кого, пока я не объясню, для чего каждый URI предназначен. Информацию из двух RDF-

документов без общих URI нельзя связать. Но два документа, у которых есть общие URI, описывают некоторые совпадающие сущности. Кто-то ещё может разместить информацию о 'Law & Order'. Если использовать одинаковые URI для телешоу в двух документах, RDF-инструменты смогут определить, что два документа описывают одно и то же.

Ниже приведён пример использования RDF для описания книг. Допустим, гипотетически, Библиотека конгресса разместила список книг в RDF и то же самое сделал Amazon.com.

[[пример: RDF данные Библиотеки конгресса (гипотетически).]]

[[пример: RDF данные Amazon.com (гипотетически).]]

URI для книг (urn:isbn:...) в этих файлах общие. RDF-приложение, используя файлы, могло бы сообщить, что 'The Future of Ideas' стоит на Amazon 9.95$, поскольку и заголовок и цена относятся к одному и тому же ресурсу, обозначенному URI urn:isbn:B00005U7WO. Если бы в файлах не использовались одинаковые URI для этой книги, ничто бы не показало, как заголовки соотносятся с ценами.

Также важно использовать те же предикаты, что и другие, если предикаты, которые вам нужны, уже существуют. Это позволит существующим приложениям использовать вашу информацию без необходимости для разработчиков их изменять, чтобы распознавать ваши собственные URI. Например, если вы описываете документы, следует использовать существующие предикаты заголовка и описания из дублинского ядра (Doublin Core, DC), чтобы RDF-приложения, уже использующие эти предикаты, могли получить заголовки и описания из ваших данных. Если вы описываете людей, применяйте существующие предикаты Friend of a Friend (FOAF), чтобы приложения, знающие FOAF, могли использовать эту информацию.

Я использовал dc:title в вышеприведённом примере с книгами, поскольку уже существует предикат для передачи информации, которую я хотел заложить в файлы — связь между книгой и её заголовком. Для указания цены на Amazon.com не существует стандартного предиката, поэтому я его выдумал. Помните: amazon:price — это сокращение от полного URI, но для краткости я опустил определение 'amazon:'.

Однако, будьте аккуратны со значением существующих предикатов. RDF-приложения ожидают, что предикат dc:title связывает документ с его заголовком. Не следует использовать его, чтобы связать, например, телефон с его номером, поскольку существующие RDF-приложения, получившие ваши данные, используют их так, что результат не будет иметь смысла.

Объединение информации: пример из жизни

Сегодня вы чаще всего получаете информацию из одного источника. Например, одна база данных может содержать информацию о линейке продуктов. В сети совсем немного распределённой информации, поскольку обычно сложно сложить её вместе, взяв из разных источников, каждый из которых может иметь свой собственный формат и соглашения.

Вот пример, в котором распределение информации имеет смысл: база данных продуктов различных производителей и обзоры продуктов от различных обозревателей. Ни один из производителей не захочет отвечать за поддержку центральной базы данных этого проекта, особенно, если она будет содержать информацию о конкурирующих продуктах и негативные отзывы о собственных. Кроме того, может быть так, что ни один обозреватель не имеет достаточно ресурсов для поддержания этой базы в актуальном состоянии. Так как же осуществить задуманное?

RDF замечательно подходит для этого проекта. Каждый производитель и обозреватель будет размещать файлы в RDF-формате на собственных сайтах. Производители выберут URI для своих продуктов и обозреватели будут использовать эти URI при написании обзоров. Производителям

не нужно согласовывать общую схему именования продуктов, а обозреватели не привязаны к формату данных производителя. RDF позволяет обоим сторонам согласовать то, что им нужно согласовать, без необходимости для кого-либо использовать некоторый общий словарь.

[[пример: Производитель №1.]]

[[пример: Производитель №2.]]

[[пример: Обозреватель №1.]]

[[пример: Обозреватель №2.]]

Остаётся открытым вопрос, как приложение получит эти файлы, но я пока его отложу. Если у приложения есть все эти файлы, у него достаточно информации, чтобы соотнести обзоры продуктов с ценами и даже с информацией, специфичной для производителя, такой как производитель1:номер_детали и производитель2:код_гарантии. То, что следует понять из этого примера — RDF очень гибок, и в то же время позволяет приложениям мгновенно связывать информацию в единое целое.

Кроме того, RDF-приложениям не нужно знать о природе данных в этих файлах, чтобы их использовать. Если приложение уже знает, для чего используются предикаты dc:title и dc:description, то оно, как минимум, может представить заголовки и тексты обзоров для продуктов. Заметьте, что присутствие предикатов, которые приложение не понимает (например, review:rating), никоим образом на него не влияет. Приложение может просто игнорировать их, без опасений неправильно истолковать остальные данные.

К тому же, производителям и обозревателям не нужно особо договариваться, чтобы осуществить это. Они должны договориться использовать RDF, но им не обязательно согласовывать какой-либо особый формат или даже конкретные URI. Было бы полезно, если бы они договорились об URI, связанных с заголовками и ценами, хотя и это не является необходимостью. Но, строго говоря, им не нужно перечислять всё, что каждый из производителей мог бы сообщить о своих продуктах. Если производитель хотел бы добавить что-то, что ещё не согласованно (номера продуктов или коды гарантии), он мог бы создать новый предикат, не нарушая существующие системы. Аналогично, обозреватели не привязаны к словарям, контролируемым производителями. Обозреватели могли бы добавлять собственные связи к своим RDF-файлам: например, рейтинги.

Можно также взглянуть на это с точки зрения возможности совместного использования информации. Формат производителя №1 может обеспечивать совместимость с любым другим форматом, несмотря на то, что производитель №1 ни с кем общий формат не обсуждал. Когда появляется кто-то, желающий обеспечить совместное использование своей информации с информацией производителя №1, ему не нужен новый формат — ему только нужно выбрать правильные субъекты, предикаты и объекты.

Это чем-то лучше XML? Я взгляну на это попозже

Сравнение RDF с XML

Ранее я показал, что RDF может быть использован для создания распределённой базы данных для информации о продуктах и обзорах. Теперь о том, как подобная система может быть реализована, при использовании XML, но не RDF.

Для начала я представлю, как это могло быть реализовано одним производителем. Производитель, если захочет, может опубликовать XML-файл с узлом для каждого продукта и дополнительными узлами внутри с именем продукта и какой-либо специфической для производителя информацией.

[[пример: XML-файл производителя №1.]]

Что мы можем сделать с этим файлом? Приложение, чтобы отобразить эту информацию, должно быть запрограммировано на знание того, что узлы <product> относятся к продуктам, заголовки описаны с помощью атрибута 'title' и т.д. И если обозреватель захочет опубликовать XML-файл с обзором, то он сможет связать его с продуктом только через имя. Два производителя могут выпускать продукты с одним наименованием, поэтому они должны будут использовать какие-то идентификаторы, чтобы различать продукты между собой.

При этом возникает первая проблема. Продавцы будут вынуждены объединиться и ввести систему идентификации продукции, чтобы идентификаторы были уникальны в локальном пространстве идентификаторов консорциума данных производителей. RDF решает эту проблему, требуя глобальной уникальности всех идентификаторов и предлагая использовать URI в качестве идентификаторов, что позволяет кому угодно использовать идентификаторы своего локального пространства, которое они контролируют.

[[пример: XML-файл продавца №1 — редакция №1.]]

[[пример: XML-файл обозревателя №1.]]

С идентификаторами в XML файлах обозреватели смогут идентифицировать продукты в своих обзорах, но приложения всё ещё не смогут соотнести продукты с обзорами. Идентификаторов не достаточно. Приложения сами по себе не смогут определить, как найти идентификаторы. В файле производителя №1 идентификатор можно найти в атрибуте id каждого узла продукта. В файлах обзоров такой же атрибут обозначает идентификатор обзора. Для установления соответствия между обзором и продуктом используется другой атрибут. Даже если производители и обозреватели договорятся, где размещать идентификаторы, приложение всё ещё будет нуждаться в информации об этой договорённости. RDF решает эту проблему присвоением каждой сущности глобального идентификатора (за исключением литералов), поэтому по существу, всё, к чему обращается RDF-приложение, — это некий идентификатор, который что-то значит.

Производители и обозреватели далее решают, что считать корректным XML-файлом описания продуктов и обзоров, и как узлы этих файлов могут интерпретироваться программным обеспечением. Если эти файлы должны соответствовать DTD или Schema, то они не могут быть произвольно расширены. Перед тем как в эти файлы будет добавлено что-то новое, например, информация, специфичная для одного из производителей, все производители и обозреватели должны будут согласовать изменение DTD или Schema.

С другой стороны, производители и обозреватели могут обойтись без DTD или Schema. Отсутствие правил, касающихся ограничений использования элементов, обеспечивает гибкость, которая нужна производителям и обозревателям. Но при этом должен быть некоторый принцип, описывающий значение элементов XML-файлов, а значит некоторый центральный полномочный орган для решения таких вопросов. Пока этого нет, производители и обозреватели используют свои пространства имён, позволяющие им разработать свои собственные словари.

Я могу продолжать, но вы уже должны были увидеть, что XML не является особенно подходящим для распределённых расширяемых данных, пока этот XML не начинает выглядеть как RDF.

RDF, описывающий RDF

До настоящего момента я показывал как можно использовать RDF для описания отношений между сущностями реального мира. RDF также может быть использован на более высоком уровне для описания RDF-предикатов и классов ресурсов. Онтологии, схемы и словари, в некотором смысле представляющие собой сходные понятия, являются RDF-данными о... других RDF-данных.

RDF-онтологии играют роль чего то в общих чертах подобного XML DTD и XML Schema. Но, насколько они похожи, настолько и отличаются друг от друга. DTD и XML Schema определяют, что собой представляет корректный документ. Они не указывают как интерпретировать документ,

и они только ограничивают набор элементов, который может быть использован. RDF-онтологии, которые сами написаны на RDF, обеспечивают отношения между сущностями верхнего уровня, полностью предназначенными для целей указания того, как приложениям интерпретировать информацию. Кроме того, онтологии не задают исчерпывающий список допустимых предикатов. Любое утверждение по-прежнему допустимо где угодно.

RDF Schema (RDFS) и Web Ontology Language (OWL) определяют несколько классов и предикатов, которые, в соответствии с соглашением, используются для обеспечения высшего уровня описания данных.

[[блок: Классы и предикаты ниже представляют собой некоторые стандартные инструменты, доступные вам для написания онтологий. Некоторые примеры полноценных онтологий, включая RDF-, RDFS- и OWL-онтологии, рассматриваются в SchemaWeb.]]

RDF Schema (RDFS)

RDF Schema (RDFS) вводит понятие класса. Класс является типом сущности. Например, вы и я являемся членами класса Person (Персона). Компьютеры являются членами класса Machine (Машины). Person и Machine — классы. Это значит, что они сами являются членами типа Class. Первый высокоуровневый предикат — это rdf:type (rdf является обычным сокращением пространства имён http://www.w3.org/1999/02/22-rdf-syntax-ns#). Этот предикат связывает некоторую сущность с другой сущностью, которая обозначает класс первой сущности. Назначение этого предиката — указать вид ресурса (обозначающего некоторую сущность). Но, как и всё в RDF, выбор класса может соответствовать соглашению или быть произвольным.

Производитель из примера, рассмотренного выше, легко может добавить информацию о классе в свои файлы:

[[пример: Добавление информации о типе.]]

Подобно выбору предикатов, полезно выбрать URI для классов, которые будут использовать другие. Согласованность разных сторон относительно классов и других вещей, рассматриваемых в этом разделе, очень важна.

Один из интересных классов — rdf:Property. Любая сущность, используемая как предикат, является rdf:Property. Таким образом, исходя из примеров выше, мы можем заключить:

[[пример: Класс rdf:Property.]]

Выражаясь конкретнее, можно включать эти высказывания в RDF-онтологию, описывающую используемые предикаты.

Другие RDFS-предикаты используются для обеспечения большей информации о предикатах. Предикаты rdfs:domain и rdfs:range связывают предикат с классами ресурсов, которые могут служить его субъектом и объектом соответственно. Например:

[[пример: Domain и Range.]]

Эти утверждения говорят о том, что субъекты предиката vendor2:warranty_code являются сущностями типа general:Product, и объекты этого предиката являются литералами. Это в самом деле так. Вспомните:

[[пример: warranty_code.]]

Сущность vendor2:product1 — это Product, а 'None.' — это литеральное значение.

Возможность задать классы допустимых субъектов и объектов для некоторого заданного предиката служит двум целям. Во-первых, это позволяет приложениям на основе утверждений делать выводы о типах сущностей. Если приложение встретит сущность, являющуюся субъектом предиката vendor2:warranty_code, может последовать вывод о том, что такая сущность принадлежит к классу general:Product. Во-вторых, эта информация служит в качестве документации для людей. То есть, RDF сам используется для демонстрации того, как предикат может быть использован.

Два RDFS-предиката используются для связи между классами и между предикатами. Отношение rdfs:subClassOf указывает, что один класс является подклассом другого. Например, класс Mammal (млекопитающее) является подклассом класса Animal (животное). Любое истинное высказывание о классе Animal также является истинным и для класса Mammal, и приложения способны сделать такой вывод при использовании такого предиката. Отношение rdfs:subPropertyOf похоже на предыдущее, но применимо к предикатам. Например, предикат friend (друг) является подсвойством для предиката knows (знакомый). Любой друг — это кто-то, кого ты знаешь.

[[блок: Смотрите документ RDF Schema, в котором подробно рассматриваются эти свойства.]]

Язык сетевых онтологий (Web Ontology Language, OWL).

Язык сетевых онтологий (OWL) определяет больше классов, что позволяет авторам RDF придавать больше смысла RDF-предикатам. OWL определяет четыре класса предикатов, включая: owl:SymmetricProperty, owl:TransitiveProperty, owl:FunctionalProperty и owl:InverseFunctionalProperty. Пространство имён OWL — http://www.w3.org/2002/07/owl#. Каждый из этих классов является rdf:subClassOf rdf:Property.

Приложения могут использовать указанные классы, чтобы, в соответствии с соглашением относительно их смысла, делать выводы относительно данных. Можно использовать эти классы в онтологиях вроде следующей:

[[пример: Определение amazon:price.]]

Поскольку указанные классы определены в OWL-онтологии как подклассы rdf:Property, приложения могут сделать следующий вывод:

[[пример: Определение amazon:price.]]

Получилось такое же утверждение, как то, что мы уже использовали раньше. Соответственно, когда вы используете подкласс на месте 'родительского' класса, вы всегда более информативны. В таком случае приложение обладает всей той информацией, что и раньше (если умеет делать выводы и знает OWL-онтологию), но кроме того получает дополнительную информацию за счёт достижения большей точности с помощью подкласса.

Симметричные свойства OWL дают приложениям сведения о том, что следующий вывод правилен: если приложение видит утверждение С-П-О, и если П — симметричное свойство, то О-П-С также верно. Например, мы считаем, что отношение 'has friend' — симметричное. Тогда, если вы — мой друг (ME HAS_FRIEND YOU), то и я — ваш друг (YOU HAS_FRIEND ME).

Транзитивные свойства OWL работают следующим образом: если приложение видит утверждение X-П-Y и Y-П-Z, и П обозначен как транзитивное свойство, то X-П-Z также верно. rdfs:subClassOf — транзитивное отношение. Если млекопитающее это подкласс животного, а животное подкласс организма, то млекопитающее подкласс организма.

Функциональные и обратно-функциональные свойства OWL показывают, сколько раз свойство может быть использовано для некоторого субъекта или объекта. Функциональное свойство то, которое имеет максимум одно значение для каждого субъекта. Например, отношение hasBirthday

между человеком и его днём рождения. У каждого есть только один день рождения, поэтому для каждого конкретного субъекта (человека) существует только один объект (день рождения). Но отношение 'обладает' между обладателем и его вещью не функциональное. Человек может обладать более чем одной вещью.

Обратные функциональные свойства работают так же, но в обратную сторону. Для любого объекта существует только один субъект с некоторым обратным функциональным свойством. Отношение has_ISBN обратно-функционально. Для любого ISBN существует только одна соответствующая ему книга. Отношение has_ISBN не может быть функциональным. Ведь может одна книга иметь более чем один ISBN!

Функциональные и обратно-функциональные свойства могут использоваться приложениями для осуществления выводов о том, что пара сущностей служат для обозначения одного и того же. Например:

[[пример: Обратно-функциональные свойства.]]

Если данные в примере согласуются между собой, тогда тот факт, что предикат ex:isbn отмечен как обратно-функциональный позволяет приложению сделать вывод о том, что имена book:a и book:b служат для обозначения одной и той же книги. Они связаны с одним и тем же ISBN и, поскольку их отношение к ISBN отмечено как обратно-функциональное, то оба субъекта должны обозначать одну и ту же книгу. Вспомним, что два имени могут ссылаться на одну и ту же сущность.

[[блок: Документ Web Ontology Language определяет семантику этих предикатов и классов.]]

Связанные данные для веба

Ранее я отметил две темы, пообещав в дальнейшем рассмотреть их подробнее. Первым было замечание о том, что при использовании 'http:'-подобных URI, напрашиваются предположения о том, что по соответствующему URL действительно что-то должно существовать. В качестве второй темы остался открытым вопрос о том, как предлагается клиентам искать данные в семантическом вебе. В сообществе вокруг семантического веба образовалось новое движение, именуемое Linked data (связанные данные), задачей которого является поиск ответов на эти вопросы.

Идея связанных данных заключается в разработке концепции по извлечению пользы от гиперссылок между документами в WWW и RDF-документами в семантическом вебе. Базовый принцип этой концепции в том, что 'http:'-подобные URI должны использоваться для RDF-ресурсов таким образом, чтобы по соответствующим URL адресам существовали RDF-документы, описывающие эти RDF-ресурсы. Когда такие документы ссылаются на другие ресурсы, если эти ресурсы имеют 'http:'-подобные URI, то клиенты семантического веба могут переходить от документа к документу, находя больше информации.

Рассмотрим пример. Допустим, URI http://www.rdfabout.com/rdf/usgov/geo/us/ny представляет штат Нью-Йорк в Соединённых Штатах. Если вы посетите этот URL, то получите RDF-документ, описывающий Нью-Йорк. И этот документ ссылается на некоторые другие ресурсы, среди которых встречаются 'http:'-подобные URI, по которым вы можете запросить документы, описывающие соответствующие ресурсы. Не путайте документы, которые вы получаете, с собственно ресурсами, обозначаемыми этим URI. Хотя нет гарантий, что полученный вами документ будет описывать ресурс, соответствующий данному URL, это предположение является некоторым ориентиром.

Терминология

Я выбрал использование 'http:'-подобных URI, поскольку они являются разыменовываемыми.

Характеристика 'разыменовываемый' имеет отношение к WWW, но не к семантическому вебу. Разыменовываемым называют URI, являющийся также URL, или, иными словами, URI начинающийся с 'http:', который, кроме всего прочего, даёт способ получения адреса и документа по этому адресу. URI, начинающиеся с 'tag:' или 'urn:' не несут информации о том, как найти соответствующие документы, поэтому такие URI не являются разыменовываемыми. Вы не можете передать их в свой браузер и получить документ. URI, которые можно указывать браузеру называются разыменовываемыми.

Разница между документом (который вы получаете через браузер) и собственно ресурсом (который именуется данным URI) ключевой момент использующейся терминологии. Электронный ресурс это такой ресурс, который может быть передан по электронным каналам связи. Документы, такие как веб-страницы, RDF/XML-документы, изображения и двоичные файлы, все они являются информационными ресурсами. Не-информационные ресурсы это такие ресурсы, которые также могут быть именованы с помощью URI, но не могут быть переданы по электронным каналам связи. Люди, абстрактные понятия и т.п. являются не-информационными ресурсами. Мы видим, что, как информационные ресурсы, так и не-информационные ресурсы могут именоваться с помощью URI. Браузер может отобразить только информационный ресурс. Хотя браузер может отобразить представление не-информационного ресурса (например, изображение человека), но он, по определению, не может отобразить сам не-информационный ресурс.

Скрытые связанные данные

Использование HTTP GET

В соответствии со стандартами архитектуры веба ответ 'HTTP 200 OK' предназначен для запросов URI, обозначающих только информационные ресурсы. Поэтому, когда вы посещаете URI города Нью-Йорк из примера выше (не-информационный ресурс), веб-сервер сначала выдает ответ HTTP 303, что означает команду ожидать другой ответ (иначе говоря, перенаправление, редирект). Это значит, что ответ на запрос данного URI сервер не может обеспечить непосредственно, поскольку не может передать не-информационный ресурс по проводам. Вместо этого сервер шлет в ответ пользователю URL, направляющий пользователя к информационному ресурсу, подразумевая, что информация о изначально запрашиваемом URI может быть найдена по указанному URL.

Если вы используете Linux, то можете наблюдать это с помощью команд curl или wget в командной строке. При запуске команды curl как показано ниже на печать будет выведен код перенаправления, получаемый браузером при посещении URI Нью-Йорка.

[[пример: Использование команды curl для получения связанных данных.]]

Если вы посмотрите внимательно, то увидите, что в этом случае перенаправление отправит вас по динамическому URL. Более подробно на этом остановимся позже. Тем не менее, часто перенаправление работает на статические RDF/XML-документы (например, с расширением .rdf).

Используйте ключ -L для перехода в соответствии с перенаправлением.

[[пример: Использование команды curl для получения связанных данных.]]

После перехода в соответствии с перенаправлением вы получите RDF/XML-документ, описывающий Нью-Йорк.

Вы также можете попробовать использовать команду wget с ключом -S для просмотра заголовков HTTP-ответа.

[[пример: Использование команды wget для получения связанных данных.]]

Согласование содержимого и теги link.

Протокол HTTP поддерживает различные виды ответных передач документов в зависимости от некоторых аспектов запроса. В частности, запрашивающий HTTP-клиент может указать в заголовке HTTP Accept желаемый MIME-тип, это может быть часто используемый HTML (text/html) или что-то другое. Это позволяет серверу переопределять свои ответы и называется согласованием содержимого.

Клиент, запросивший через веб-браузер некоторый RDF URI может требовать получения HTML-представления ресурса для его отображения пользователю (некоторое текстовое описание ресурса, обозначенного с помощью запрашиваемого URI). В то же время, RDF-клиент может требовать RDF/XML описания ресурса. В этом случае клиент указывает в заголовке Accept следующее: application/rdf+xml. Получается, что HTML-страница и RDF/XML-документ являются двумя информационными ресурсами, представляющими один и тот же не-информационный ресурс (который обозначается с помощью запрашиваемого URI).

HTTP GET запросы RDF URI, которые мы только что обсуждали, являются для RDF-клиентов одним из путей для нахождения триплетов об отдельных ресурсах. Согласование содержимого позволяет получать HTML-страницу, либо RDF-документ по одному и тому же URL. Другая возможность связать HTML-страницу с RDF-документом заключается в использовании HTML тега link. Этот тег помогает RDF-клиентам найти RDF-информацию, связанную со страницами, которые просматривает пользователь.

Размещение тегов link в заголовке HTML-страниц:

[[пример: Тег link.]]

В этом примере файл moreinfo.rdf является RDF/XML-документом, и, как следует полагать, либо описывает ресурс, соответствующий URI страницы, на которой располагается тег link, либо описывает тот ресурс, которому посвящена эта HTML-страница.

[[блок: Для получения более подробной информации о Linked Data и лучших практиках применения данной технологии ознакомьтесь с документом 'Как публиковать связанные данные в веб'.]]

Запросы к базам данных семантического веба

Базы данных семантического веба или RDF-базы, также называемые хранилищами триплетов, поскольку в них хранятся триплеты (которые, если вы вспомните, иначе называются RDF-высказываниями), это обычно произвольные хранилища RDF-высказываний, поддерживающие некоторые виды запросов. Естественно, важно, имея документ, иметь возможность запросить информацию о его содержимом.

Механизм запроса информации из хранилища абстрагирован от того, в каком формате данные были до размещения в хранилище (например, в XML или нотации 3). В результате, в хранилище часто нет информации о порядке, в котором высказывания следовали друг за другом в исходном документе, о том, какое использовалось пространство имен, префикс и локальное имя для того или иного ресурса, а также о том, какие локальные имена использовались для анонимных узлов. Все это особенности сохранения RDF в последовательной форме, не являющиеся частью информационного содержимого документов, и эти особенности, соответственно, следует игнорировать.

Хранилище триплетов предназначено для выдачи информации о том, какие высказывания в нем содержатся (в неупорядоченном виде). Обычно хранилище может ответить на запрос о наличии конкретного заданного высказывания, а некоторые могут выдать набор высказываний, соответствующих заданному шаблону.

Все хранилища триплетов грубо могут быть поделены на два типа. Один из типов низкоуровневые хранилища, обеспечивающие возможность ответа на вопрос о наличии триплета. Хранилища другого типа позволяют получать ответы на более сложные запросы, связанные с семантикой RDFS и OWL (рассмотренной нами выше) или с другими специально поддерживаемыми логическими правилами. Мы будем говорить только о хранилищах первого типа.

[[блок: В качестве списка реализованных хранилищ триплетов смотрите LargeTripleStores на ресурсе ESW wiki.]]

Аналогично системам управления реляционными базами данных и XML, RDF тоже обладает стандартным языком запросов, который называется SPARQL. Его лексикографическое подобие языку SQL не случайно. SPARQL, язык запросов для RDF, в общих чертах подобен SQL, хотя в целом они различны. Более того, стандарт SPARQL имеет отношение только к запросам информации. SPARQL не имеет никакого отношения к добавлению высказываний в хранилище триплетов или редактированию содержимого хранилища. Хранилища триплетов могут поддерживать подобные операции, но не на основе стандарта SPARQL.

Аналогично рассмотренным выше введениям в нотацию 3 и RDF/XML, цель этой главы демонстрация основ языка SPARQL.

Давайте начнем с некоторых примеров наборов данных в гипотетическом RDF-хранилище.

[[пример: RDF-данные о моей квартире.]]

В языке SPARQL предусмотрено четыре вида запросов: SELECT, ASK, DESCRIBE и CONSTRUCT.

Запрос SELECT наиболее близок к SQL. Ответ на этот запрос предоставляется в табличной форме. На этом сходство заканчивается. С помощью SPARQL-запроса SELECT вы можете осуществлять поиск ресурсов. Например, можно запросить ресурс, который 'владеет' (предикат 'own') 'моей_квартирой' (ресурс 'my_apartment'). В ответе будут указаны ресурсы x, о которых в хранилище присутствуют высказывания в форме x ex:own taubz:my_apartment. Ответ, соответствующий приведённому выше примеру будет taubz:me. Другой пример запроса: какие ресурсы находятся в моей квартире? Этот запрос формулируется так: какие ресурсы являются объектами x в триплетах, соответствующих шаблону taubz:my_apartnemtn ex:contains x. В этом отличие от SQL, в котором каждая строка результата является строкой таблицы. Здесь каждая строка ресурс.

В SPARQL, как и в SQL, есть ключевое слово WHERE. Часть запроса, начинающаяся с WHERE, как и всё в SPARQL, является шаблоном графа. Шаблон графа, как и граф в формате нотации 3, может включать в себя URI, анонимные узлы, литералы, кроме того, допускается использовать переменные. Переменные служат точно так же, как использованный нами выше x. Переменные обозначают те ресурсы, которые вы ищите. Второй запрос из рассмотренных выше можно записать так:

[[пример: 1-й пример SPARQL запроса.]]

Результаты будут такими:

[[пример: Результаты 1-го примера SPARQL запроса.]]

Теперь несколько замечаний. Во-первых, выбор имени переменной ?what произволен, аналогично именам в самом RDF, за исключением того, что в SPARQL имена переменных начинаются со знака вопроса. Во-вторых, хотя некоторые вещи, например, шаблон графа, выглядят вполне соответствующими синтаксису N3, другие ему не соответствуют, такие как объявление PREFIX. И

последнее замечание, результат запроса с математической точки зрения представляет собой таблицу. Механизм, обеспечивающий доступ к этой таблице, зависит от используемого движка SPARQL. Как минимум, для этой цели подходит стандартный XML-формат вывода данных. Не будем сейчас заострять внимание на этом.

Шаблон графа может содержать более одного высказывания. Это имеет следующее значение: 1) фильтрация возвращаемых ресурсов или 2) запрос базируется на длинных цепочках данных в репозитории. Например, в запросе о тех ресурсах в моей квартире, которыми я владею, второе высказывание ограничивает список возвращаемых ресурсов только теми, которыми я владею:

[[пример: 2-й пример SPARQL запроса.]]

Результаты будут такими:

[[пример: Результаты 2-го примера SPARQL запроса.]]

Каждый возвращённый ресурс должен подставляться одновременно во все вхождения соответствующей переменной ?what таким образом, чтобы каждое утверждение шаблона графа существовало в хранилище. Ресурс taubz:барахло_моих_друзей не соответствует этому запросу, поскольку при подстановке этого ресурса в шаблон графа получается:

[[пример: Шаблон графа с подстановкой.]]

одно из этих утверждений, второе по счёту, не существует в хранилище.

Запрос по поиску ресурсов, 'содержащихся в' любом объекте, которым я владею (это мой компьютер в моей квартире, барахло моих друзей также в моей квартире, ручки и карандаши в моем столе) представляет собой шаблон графа, использующий длинную цепочку ресурсов в хранилище. Мы можем переформулировать этот запрос как конъюнкцию двух утверждений, как и прежде, но в этот раз содержащих две переменные. А именно: для любой найденной пары x и y должно выполняться оба утверждения: A) taubz:me ex:own y и B) y ex:contains x. SPARQL-запрос и ожидаемые результаты следующие:

[[пример: 3-й пример SPARQL запроса.]]

[[пример: Результаты 3-го примера SPARQL запроса.]]

Усложним пример, запросив пары ресурсов ?container и ?what вместо списка искомых объектов. Изменения заключаются в добавлении переменной ?container непосредственно после слова SELECT:

[[пример: 4-й пример SPARQL запроса.]]

[[пример: Результаты 4-го примера SPARQL запроса.]]

Каждая строка результата проверяется подстановкой каждой переменной из соответствующей строки результата в шаблон графа, причём все переменные из одной строки подставляются одновременно. Как и прежде, если после подстановки всех переменных в шаблон графа все получившиеся высказывания присутствуют в хранилище, то такой набор переменных включается в возвращаемый результат.

Немного терминологии перед тем как мы закончим. Относительно каждой отдельной строки результата переменную называют 'связанной' с соответствующим значением. В первой строке таблицы результатов для 4-го примера, рассмотренного нами выше, переменная ?container связывается со значением taubz:my_apartment.

На этом завершаем рассмотрение SPARQL. По крайней мере до выхода очередной версии этой статьи.

[[блок: Ознакомьтесь со спецификацией SPARQL для дальнейшего знакомства с языком запросов.]]

Заключительные замечания

Кое что представляет собой уже тот факт, что кто-то сможет говорить о RDF, не будучи вынужденным при этом рассказывать все то, что уже изложено в этой статье. Я надеюсь, что эта статья была полезной. Комментарии приветствуются.

В этой статье мы ссылаемся на другие хорошие начинания, касающиеся RDF.

Если вы хотите программировать, используя RDF, то пользуйтесь множеством инструментов, информация о которых и сами программы переведены на многие языки.

Огромная благодарность каждому участнику списка рассылки W3C по семантическому вебу, кто отреагировал на первоначальную публикацию этой статьи!

Translation: © Владимир Соколовский, daeq, jupy, englishextra, Александр Островский, Дмитрий Уланов, Igor Artamonov, amabilis, tiex, dionys.myopenid.com